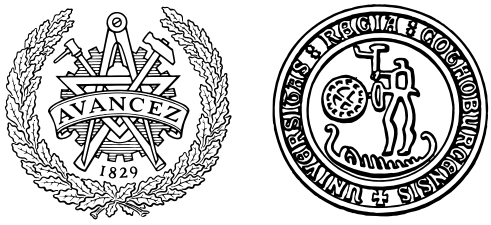
Thesis for the Degree of Licentiate of Philosophy

A Richardson Leja Lanczos Algorithm to Compute Interior Eigenvalues of Very Large Matrices

Sonia Gupta

CHALMERS | GÖTEBORGS UNIVERSITET



Department of Mathematics
Chalmers University of Technology and Göteborg University
Göteborg, Sweden 2001

A Richardson Leja Lanczos Algorithm to Compute Interior Eigenvalues of Very Large Matrices
Sonia Gupta
ISSN 0347-2809
NO 2001:18

Department of Mathematics
Chalmers University of Technology and Göteborg University
SE-412 96 Göteborg, Sweden

Abstract

An algorithm is developed for the numerical computation of eigenvalues in a selected interval inside the spectrum of a large sparse matrix.

It uses the Lanczos algorithm with a filtered starting vector. The unwanted directions are filtered out using the Richardson iteration with the Leja shifts covering the part of the spectrum where no eigenvalues are wanted. Leja points have a similar distribution as zeros of Chebyshev polynomial, but have the advantage that new Leja points can be added to a given set of Leja points. Two variants of the algorithm, single vector Lanczos and block Lanczos are developed to find all the desired eigenvalues in any specified part of the spectrum.

The main advantage of the algorithm is that no factorization of a large matrix is needed, Richardson iteration can be performed without reorthogonalization and very few Lanczos iterations are needed.

Numerical results are reported on two different classes of matrices. The first one is obtained from the model used for the investigation of electronic behaviour of disordered system in the quantum physics. The second set of matrices arise in physical chemistry while studying the molecular excited state. The results are compared with the Lanczos algorithm by Cullum and Willoughby.

Key words: Lanczos algorithm, Block Lanczos, Orthogonalization, Richardson iteration, Leja points, Chebyshev polynomial.

2000 MSC: 65F15, 65F50

Acknowledgement

I would like to express my sincere gratitude to my advisor Prof. Axel Ruhe for his constant support, encouragement and guidance. In spite of his hectic schedule, he always spared his valuable time to meet me, discuss the queries that I had and read and comment on the draft innumerable number of times.

I wish to thank Dr. Thomas Ericsson for his enthusiasm, enlightening discussions and suggestions. His influence is felt throughout the work. He not only helped me in working with the computer but also introduced me to many important concepts in Numerical Analysis.

It is my pleasure to share the office with Katarina. I am indebted to Mia for her unconditional friendship. Finally, special thanks to my friends both from the world in and outside the mathematics.

To my parents!

Contents

1 Introduction

In the world of matrix computation, many algorithms exist which deal with numerical computation of eigenvalues λ and eigenvectors x of a matrix A given by

$$Ax = \lambda x.$$

If the size of the matrix A is moderate then there are many efficient algorithms, see for example Golub and Van Loan [11], Bai *et al.* [3], Demmel [7].

In many applications, the matrix A is large, sparse and symmetric and we need to compute only few eigenvalues and eigenvectors. Extensive research has been done when matrix is symmetric and sparse. Algorithms for such matrices is treated in Parlett [15].

Algorithms based on the Lanczos method are standard choice when the size of the matrix is large. The main asset of the Lanczos method is that we only need matrix vector multiplication. But, as Paige [14] pointed out in his thesis, the Lanczos vectors start loosing orthogonality when an eigenvalue starts to converge. A trivial remedy proposed was to reorthogonalize the Lanczos vectors in each step. This is extremely costly both in terms of computation and storage. The other extreme, no reorthogonalization is advocated by Cullum and Willoughby in [6]. This is an efficient technique as long as only extreme eigenvalues are needed which are neither clustered nor multiple. Cullum and Willoughby's algorithm run Lanczos algorithm for many iterations, and so we have to compute eigenvalues of a symmetric tridiagonal matrix which may be larger than the size of the original matrix. Parlett [15] proposed selective reorthogonalization, which gives almost the same accuracy as full reorthogonalization techniques in the computed eigenvalues, with reduced computational and storage cost.

Orthogonality among Lanczos vectors is lost as soon as Ritz vectors becomes a good approximation of an exact eigenvector of the matrix A. According to Paige's result [15, pg. 256], we should orthogonalize a Lanczos vector against the good Ritz vectors when the cosine of the angle between the Lanczos vectors become greater than the square root of the machine precision. Hence, selective reorthogonalization ensures that the tridiagonal matrix obtained from the Lanczos algorithm is an accurate projection of

matrix A on the subspace spanned by the Lanczos vectors even when they are orthogonal only to the square root of the machine precision.

Apart from the storage requirement for the Krylov subspace generated and the computational cost to preserve the orthogonality among the computed Krylov subspace basis, the Lanczos algorithm gives poor convergence to interior eigenvalues. For any random starting vector, the Lanczos algorithm always converges to end eigenvalues first. It fails to give good convergence to eigenvalues in the interior of the spectrum, unless it is combined with a shift and invert spectral transformation. Such an algorithm for few non-extreme eigenvalues is described by Ericsson and Ruhe [10], Parlett [15]. If the size of the matrix is very large and it is sparse then the computational effort required for the factorization in the shift and invert will decrease the efficiency of the algorithm. Also, storage requirements for the factors may make such algorithms unattractive to use.

This report presents a new iterative method for the computation of a few selected eigenvalues of a large sparse symmetric matrix A in any part of the spectrum. This method does not require the factorization of the matrix A. The main idea of the algorithm is to produce a starting vector for the Lanczos algorithm in the invariant subspace associated with the desired eigenvalues. This is done by using Richardson iteration with shifts. Richardson iteration is well known for solving system of linear equations and it is known that optimum choice of shifts are reciprocal of the zeros of the Chebyshev polynomial, described by Young in [20]. In our case, we use Richardson iteration with shifts to purify the starting vector for the Lanczos run.

Suppose we are interested in the m smallest eigenvalues of the matrix $A \in \mathbb{R}^{n \times n}, n >> m$. We start our algorithm by taking any random vector. We choose k shifts in an interval $\mathbb{K}$ on the real axis which contains the whole spectrum of A except for an interval containing the m desired eigenvalues. When Richardson iteration with these shifts is applied to the vector, we get a new vector which has large components outside the interval $\mathbb{K}$ and small components inside $\mathbb{K}$. This means we get a filtered initial vector having large components in the directions corresponding to the m wanted eigenvalues. In other words, Richardson iteration with shifts amplify the components of a random starting vector in the direction corresponding to all

the eigenvalues outside the region where shifts are chosen.

It is not easy to decide beforehand how many shifts will be needed to suppress undesired directions in the given starting vector. So it may be necessary to add few more shifts to the already computed sequence of shifts. If shifts are zeros of the Chebyshev polynomial then it is not easy to update the computed sequence. This is due to the fact that the Chebyshev polynomial of degree k and $k+1$ do not have common zeros. Because of this difficulty, Calvetti *et al.* use Leja points as shifts in Richardson iteration, both in eigenvalue computation [5, 1] and also for solving systems of linear equations [4]. In section 3, we will discuss Leja points and its variant fast Leja points in detail.

Leja points and zeros of the Chebyshev polynomials over the same interval are distributed almost identically. The main advantage of Leja points is that it is easy to add new Leja point successively to an already computed sequence of Leja points.

The next step in the algorithm is to use the vector obtained from Richardson iteration as the starting vector for the Lanczos iteration. Since, all directions, except for the ones associated with the wanted eigenvalues, are suppressed, we get convergence to all the desired eigenvalues after very few steps.

Hence, the main features of the algorithm includes no factorization of the large sparse matrix A, Richardson iteration without reorthogonalization and very few Lanczos iterations.

The need for such an algorithm was motivated by a physical chemistry application. While studying the molecular excited states, one is interested in eigenvalues (energy levels) several levels above the lowest eigenvalue (ground state). The matrix obtained is very large and sparse. The test matrix provided to us is of size more than 100,000 which is 0.14% filled. It is not feasible to find the interior eigenvalues of such large matrix using the shift and invert transformation technique. The Cullum and Willoughby technique may provide us with a tridiagonal matrix which is much larger than the original matrix.

The rest of the report is organized as follows. Some notations and

definitions are explained in section 2. Section 3 discusses the Richardson iteration and fast Leja points in detail. The main algorithm along with some variants is the main topic for section 4. We discuss implementation aspects of the algorithm briefly in section 5. Section 6 is devoted to some illustrative numerical examples.

2 Notations and Definitions

The *eigenvalues* of a matrix $A \in \mathbb{R}^{n\times n}$ are the n roots of its characteristics polynomial $p(z) = det(zI - A)$. The set of these roots is called the *spectrum* and is denoted by $\lambda(A)$. If $\lambda \in \lambda(A)$, then the nonzero vectors $x \in \mathbb{C}^n$ satisfying

$$Ax = \lambda x$$

are referred to as *eigenvectors*.

When A is symmetric, i.e. $A = A^T$, all the eigenvalues are real and there is an orthonormal basis of eigenvectors. Eigenvalues of any symmetric matrix A satisfy following theorem by Weyl:

Theorem 1 [7, pg. 198] *Let A and E be $n \times n$ symmetric matrices. Let $\lambda_1 \geq ... \geq \lambda_n$ be the eigenvalues of A and $\hat{\lambda}_1 \geq ... \geq \hat{\lambda}_n$ be the eigenvalues of $\hat{A} = A + E$. Then*

$$|\lambda_i - \hat{\lambda}_i| \leq ||E||_2.$$

We can get a tighter bound on approximate eigenvalues if they are approximated using the Rayleigh quotient.
The *Rayleigh quotient* of a symmetric matrix A and nonzero vector u is

$$\rho(u, A) = (u^T Au)/(u^T u).$$

Theorem 2 [7, pg. 205] *Let A be symmetric and x be a unit vector. Let $r = Ax - \rho(x, A)x$ and λ_i be the eigenvalue of A closest to $\rho(x, A)$. Let $gap \equiv \min_{j\neq i}|\lambda_j - \rho(x, A)|$. Then*

$$|\lambda_i - \rho(x, A)| \leq ||r||_2^2/gap.$$

This theorem gives a base for using the Rayleigh-Ritz procedure to approximate the eigenvalues of A when using the Lanczos algorithm.

With the Lanczos method, we build up an orthogonal basis V_m of the Krylov subspace,

$$K^m(A, v) = \text{span}\{v, Av, ..., A^{m-1}v\}$$

with v as a starting vector.
The vectors in V_m satisfies a three term recursion

$$AV_m = V_m T_m + r_m e_m^T$$

where $r_m = v_{m+1}\beta_m$, T_m is the real symmetric tridiagonal matrix,

$$T_m = \begin{bmatrix} \alpha_1 & \beta_1 & & & & & \\ \beta_1 & \alpha_2 & \beta_2 & & & & \\ & \ddots & \ddots & \ddots & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & \beta_{m-2} & \alpha_{m-1} & \beta_{m-1} \\ & & & & & \beta_{m-1} & \alpha_m \end{bmatrix} \tag{1}$$

We approximate the eigenvalues of A by the eigenvalues of $T_m = V_m^T A V_m$. These approximations are called Ritz values and are denoted by θ_i. The corresponding approximate eigenvector is the Ritz vector defined as $x_i = V_j s_i$ where s_i is an eigenvector of T_m.

Instead of using a single vector as a starting vector it is possible to use a block of orthogonal vectors. This block version of the algorithm is called block Lanczos [12]. If $V \in \mathbb{R}^{n\times p}$ is a starting orthonormal block then the Krylov subspace associated with V is

$$K^m(A, V) = \text{span}\{V, AV, ..., A^{m-1}V\}.$$

As in the case of single vector, using Lanczos algorithm an orthonormal basis for the Krylov subspace is obtained. After m steps, we get a basis V_m satisfying a three term recursion

$$AV_m = V_m T_m + R_m E_m^T$$

where $R_m = V_{m+1}B_k$, $E_m^T = (0, 0, ..., I_p)$ and T_m is the real symmetric block tridiagonal matrix,

$$T_m = \begin{bmatrix} M_1 & B_1^T & & & & & \\ B_1 & M_2 & B_2^T & & & & \\ & \ddots & \ddots & \ddots & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & B_{m-2} & M_{m-1} & B_{m-1}^T \\ & & & & & B_{m-1} & M_m \end{bmatrix} \tag{2}$$

where $M_i \in \mathbb{R}^{p\times p}$ and $B_i \in \mathbb{R}^{p\times p}$ is upper triangular.

3 Richardson iteration and Fast Leja points

Richardson iteration

Richardson iteration is a well known iterative method for solving a system of linear equations

$$Ax = \overline{b}$$

where x and $\overline{b}$ are in $\mathbb{R}^n$. It computes a sequence of vectors $x_1, x_2, ..., x_n$ where

$$x_{k+1} = x_k + \delta_k(Ax_k - \overline{b}), \quad k = 1, 2, ...$$

where $\delta_k \in \mathbb{R}$ is a sequence of shifts. Let eigenvalues of A lie in $[a, b] = \lambda(A)$. It is shown by Young [20, pg. 361–365] that the good choice for δ's are the reciprocal of the zeros of the Chebyshev polynomial in the interval $[a, b]$.

Suppose we have an eigenvalue problem

$$Ax = \lambda x$$

where the spectrum of A lies in the interval $\lambda(A) = [a, b]$. We are interested in the m smallest eigenvalues of A which are lying in the interval $[a, a_1]$ which is a subset of $\lambda(A)$. We can use Richardson iteration with shifts to generate a good starting vector where components in the unwanted directions are suppressed.

If q_0 is a starting vector then using Richardson iteration

$$q_{k+1} = (A - \zeta_k I)q_k, \quad k = 1, 2, ... \tag{3}$$

with shifts ζ_k, we try to improve the starting vector for all eigenvalues in the interval $[a, a_1]$. To achieve this we choose shifts in $\mathbb{K} = [a_1, b]$. The best shifts will be the eigenvalues of the matrix A lying in the interval $\mathbb{K}$. But in all practical situations eigenvalues of a given matrix is never known. Hence, taking the idea from system of linear equations, an obvious choice for the shifts will be the zeros of Chebyshev polynomial of degree k in the interval $\mathbb{K}$.

Suppose the starting vector obtained after applying k shifts is not good and we want to interpolate at $k+1$ zeros of Chebyshev polynomial of degree $k+1$. It is difficult to take advantage of previous k shifts since the Chebyshev polynomial of degree k and $k+1$ do not have common zeros. This motivate us to use shifts obtained from some other polynomial which is computationally efficient and also retains the good properties of the Chebyshev polynomial. A good replacement was proposed by Calvetti *et al.* [4] where they use Leja points as shifts for Richardson iteration for solving the system of linear equations.

Leja points

Leja points were studied by Edrei [8] and Leja [13] for a compact set $\mathbb{K}$ in the compact plane $\mathbb{C}$ with a connected complement. Any sequence of points which satisfies the conditions

$$|\zeta_1| = \max_{\zeta \in \mathbb{K}} |\zeta|, \zeta_1 \in \mathbb{K}, \tag{4}$$

and

$$\prod_{i=0}^{k-1} |\zeta_k - \zeta_i| = \max_{\zeta \in \mathbb{K}} \prod_{i=0}^{k-1} |\zeta - \zeta_i|, \quad \zeta_k \in \mathbb{K}, \quad k = 2, 3, ... \tag{5}$$

are called Leja points for $\mathbb{K}$. The points ζ_k obtained from (4) and (5) might not be unique. In our case, set $\mathbb{K}$ is a union of one or more closed and bounded intervals on the real axis.

The set $\mathbb{K}$ is chosen so that it contains none of the desired m eigenvalues and all or most of the undesired eigenvalues. By choosing shifts for

the Richardson iteration as Leja points for such a set, we damp eigenvector components associated with undesired eigenvalues in the initial Lanczos vector.

In Reichel [16], it is shown that the products $\prod_{i=0_{i\neq k}}^{k-1} |\zeta_k - \zeta_i|$ and $\prod_{i=0}^{k-1} |\zeta - \zeta_i|$ may grow or decrease exponentially with k. So, large k can give rise to overflow or underflow while computing the Leja points. To avoid such difficulty we scale $\mathbb{K}$ to have length 4, or $\mathbb{K} = (b-a)/4$ should be approximately equal to 1; see Baglama *et al.* [2] or Reichel [16] for details. In figure 1 we show the Leja points distribution in two intervals of different length. In the interval $[-2, 2]$ whose length is 4, computing 5000 Leja points is no problem. However, when the interval is $[-10, 10]$ then after computing 400 Leja points we observe that points computed are not evenly distributed. This is caused by having an interval of length greater than 4 which gives rise to overflow.

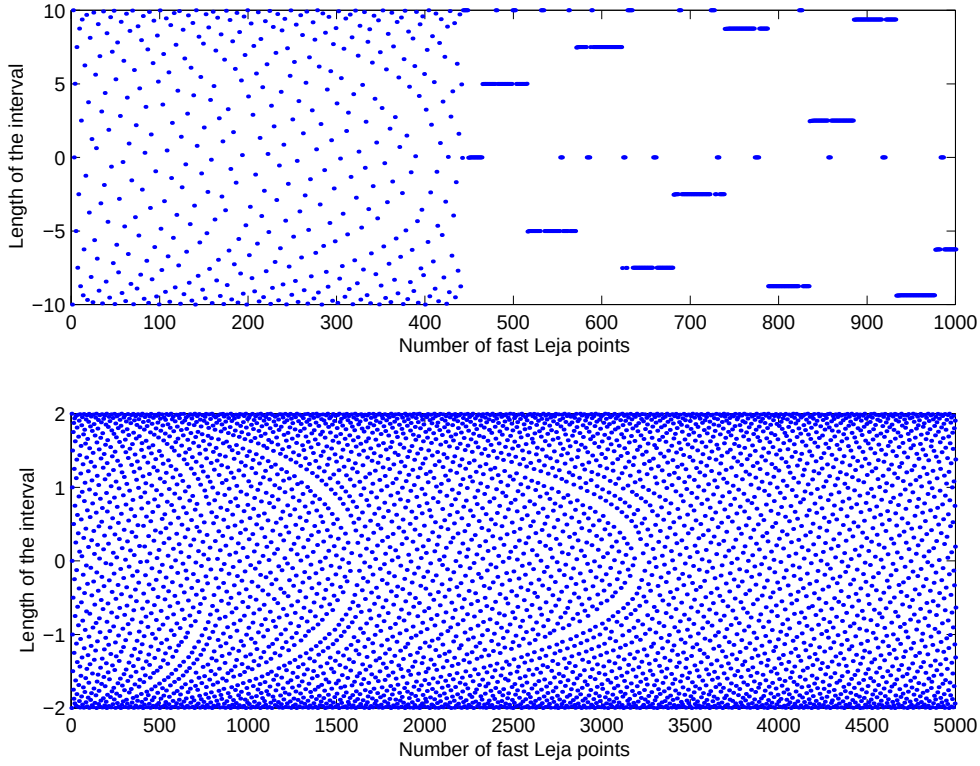


Figure 1: Leja points generation when the length of the interval is 4 and more than 4.

If a large number of Leja points is required then taking the maximum of a sequence of products can be a cumbersome work. Baglama, Calvetti and Reichel [2] introduced a new set of points, called fast Leja points, which are simpler and faster to compute than Leja points.

Fast Leja points

Fast Leja points are defined by taking the maximum in (5) not over the whole set $\mathbb{K}$ but over the finite set of candidate points c_j's which are set of points in $\mathbb{K}$ computed dynamically during the computation. The fast Leja points satisfy a condition

$$\prod_{i=0}^{k-1} |\zeta_k - \zeta_i| = \max_{0 \le j \le k} \prod_{i=0}^{k-1} |c_j - \zeta_i|, \tag{6}$$

where $\{\zeta_j\}_{j=1}^{\infty}$ are fast Leja points and k is the number of fast Leja points. Between two consecutive fast Leja points there is only one candidate point.

Suppose we want a sequence of fast Leja points $\{\zeta_j\}_{j=1}^{k}$ in $\mathbb{K}$ consisting of only one interval $[a_1, b]$. We start our computation by first defining ζ_1 and ζ_2. The first two fast Leja points are endpoints of the interval, i.e. $\zeta_1 = a_1$ and $\zeta_2 = b$. Using the condition (6) we find next fast Leja point ζ_{k+1} from the set of candidate points c_j. After determining the new fast Leja point, we remove this point from the set of candidate points. New candidate points are introduced between the new fast Leja point and the closest fast Leja points. This guarantee that the new set of fast Leja points and the new set of candidate points interlace.

As as example, let $\mathbb{K} = [-2, 2]$. Define the first two fast Leja points as $\zeta_1 = -2$ and $\zeta_2 = 2$. Then the first candidate point $c_1 = (\zeta_1 + \zeta_2)/2 = 0$. The next fast Leja point is $\zeta_3 = c_1$. We reuse c_1 to store next candidate point. The new candidate points are $c_1 = (\zeta_3 + \zeta_1)/2 = -1$ and $c_2 = (\zeta_2 + \zeta_3)/2 = 1$. We choose ζ_4 to be either c_1 or c_2. Now we update the set of candidate points by introducing two new candidate points between ζ_4 and fast Leja points closest to ζ_4. From the new set of candidate points the point which satisfies the condition (6) is a new fast Leja point.

MATLAB code to generate fast Leja points in one interval is given in [2]. With little modification, fast Leja points can be generated when $\mathbb{K}$

is a union of several intervals. Figure 2 shows the distribution of fast Leja points for different sets $\mathbb{K}$. The set $\mathbb{K} = [-2, 2]$ in the top figure. In the middle and last figures, $\mathbb{K}$ is a union of two and three intervals respectively, which are namely $[-2, -1] \cup [1, 2]$ and $[-2, -1] \cup [-0.5, 0.5] \cup [1, 2]$.

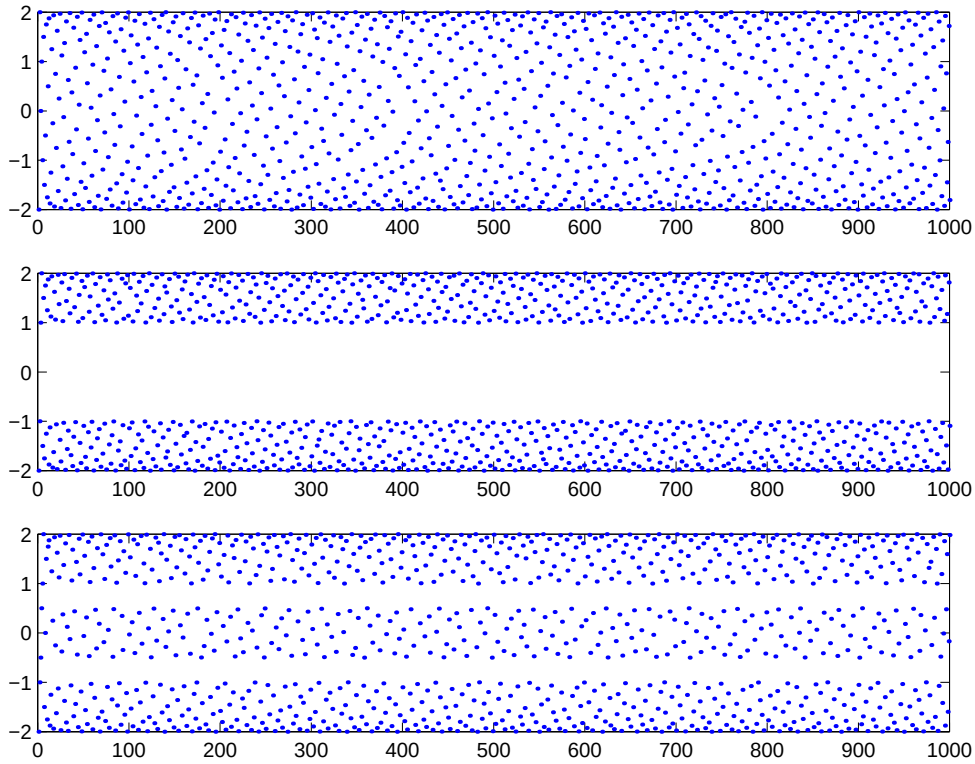


Figure 2: Distribution of 1000 fast Leja points in different intervals. Top: $[-2, 2]$,middle:$[-2, -1] \cup [1, 2]$,last:$[[-2, -1] \cup [-0.5, 0.5] \cup [1, 2]$.

Figure 3 shows zeros of Chebyshev polynomial and fast Leja points in the interval $[-2, 2]$. The degree of the Chebyshev polynomial and the Leja polynomial is 25, 50 and 100 in the top, middle and last figures respectively. From these figures we observe that in same interval zeros of the Chebyshev polynomial and the fast Leja points are distributed almost identically.

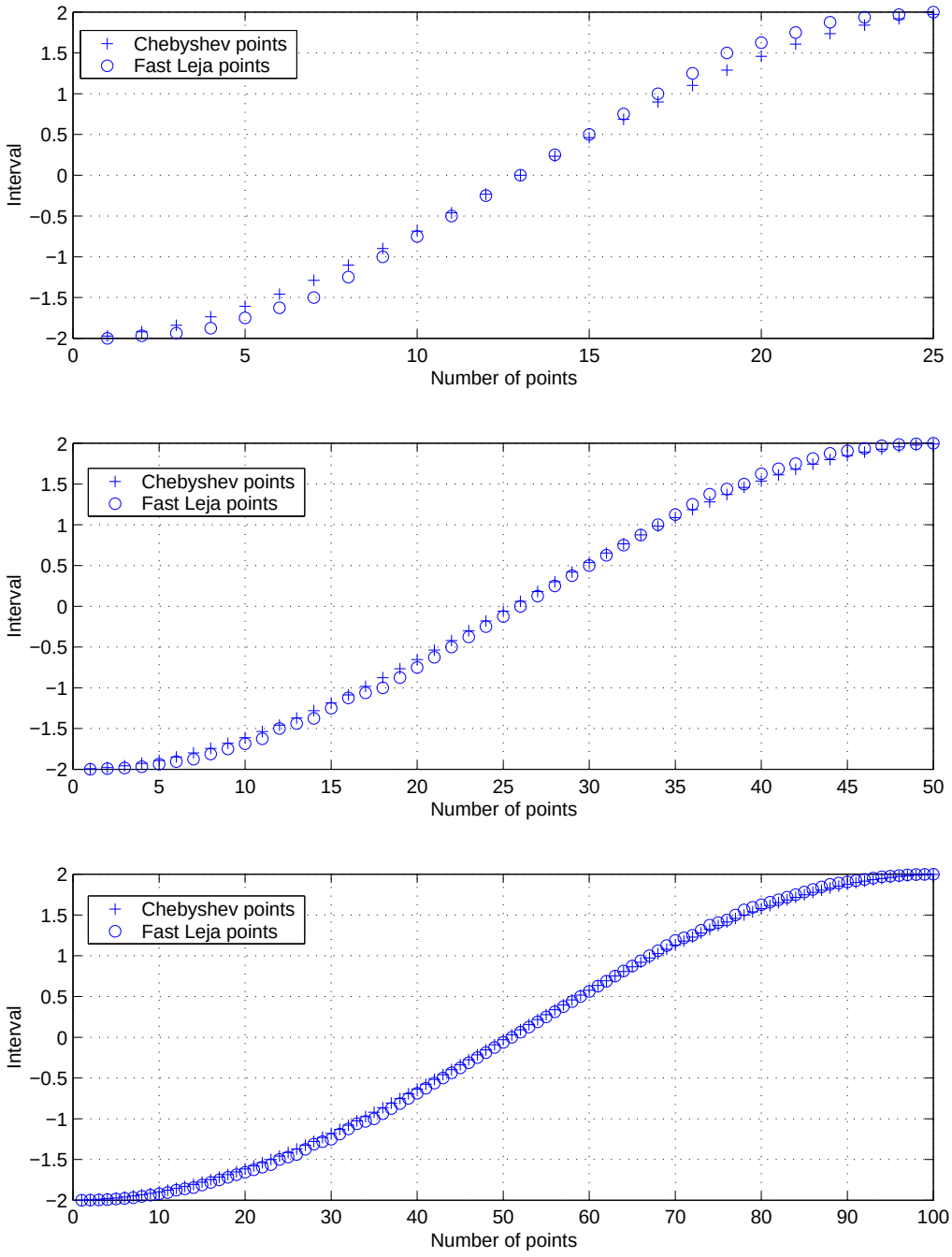


Figure 3: Comparison of zeros of the Chebyshev polynomial and the fast Leja points in the interval $[-2, 2]$.

We take these points as shifts in the Richardson iteration (3) and apply on a random starting vector whose elements have a normal distribution. The one such vector is shown in figure (4, top). We take $A = diag(a_1, a_2, ..., a_{100})$ where a_i's are equidistant points in $[-3, 2]$. The first 20 eigenvalues of A lie in A $[-3, -2)$. In figure 4, we see the effect of the Richardson iteration with 50 and 100 shifts on the random vector. Since we have taken the zeros of the Chebyshev polynomial and fast Leja points in the interval $[-2, 2]$, we observe that the Richardson iteration with these as shifts has dampened the components in the starting vector corresponding to the 80 eigenvalues of A lying in $[-2, 2]$. The first 20 components of the vector are large because they correspond to the 20 eigenvalues of A where no shifts are computed. For a general matrix A, we should plot the absolute value of $x^T q_0$ where x is an eigenvector of A and q_0 is a random starting vector.

From these figures, it is evident that either zeros of the Chebyshev polynomial or fast Leja points can be used as shifts in Richardson iteration. But in the computation of eigenvalues it is not known a priori how many iterations are required to get the desired accuracy for the wanted eigenvalues. We use the Leja points as shifts since it is easy to update k Leja points to $k + 1$ Leja points.

To achieve good performance from the Richardson iteration the order in which we apply the Leja points as shifts is important. In Reichel [16] , it is shown by numerical example that to avoid propagated round-off error we should use Leja points in the order they are generated.

4 Algorithm

The basic idea of the algorithm is to find a good starting vector for the Lanczos algorithm using the Richardson iteration with fast Leja points as shifts. As an effect of this, we filter the starting vector in the direction of desired eigenvectors. We present three algorithms for computing the end eigenvalues and the interior eigenvalues of a given matrix. These variants are motivated by the numerical experiments. The choice of the algorithm depends on the spread of eigenvalues and the need of the user.

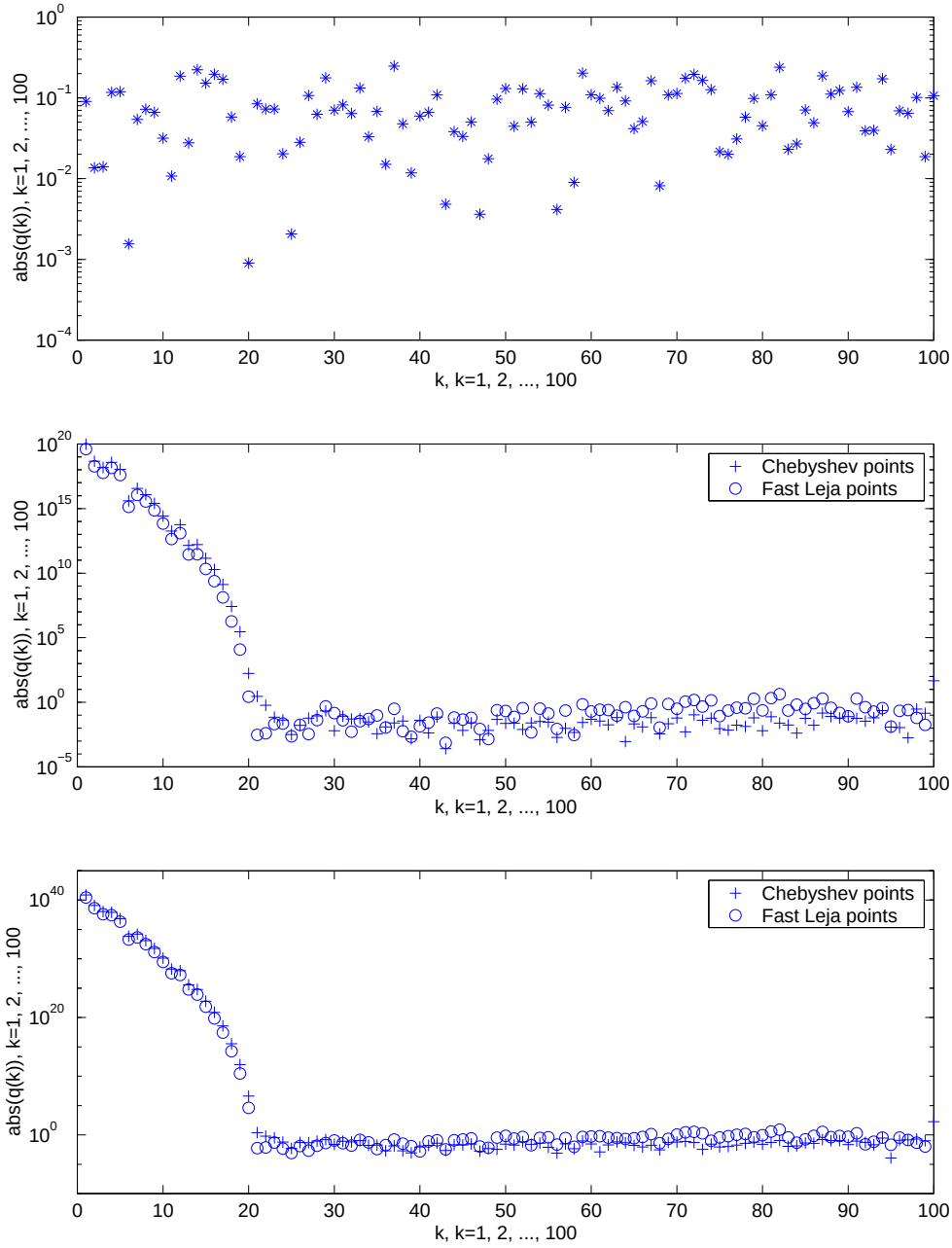


Figure 4: Top: Normalized random vector. Middle and last: Effect of Richardson iteration with zeros of the Chebyshev polynomial and the fast Leja points as shifts on the above vector. The vector in the middle figure is obtained with 50 shifts and the last figure with 100 shifts.

Algorithm 1: This is a single vector variant which performs well when well separated end eigenvalues are wanted.

1. *Input:* End points of the interval $\mathbb{K}$, number of fast Leja points k, number of Lanczos steps m, normalized starting vector q_1.

2. *Richardson iteration*
   for $j = 1, 2, 3, ..., k$
   Compute ζ_j fast Leja point in $\mathbb{K}$.
   Set $q_{j+1} = (A - \zeta_j I)q_j$
   end

3. *Lanczos run*
   $v_1 = q_k/\|q_k\|_2$
   $\beta_0 = 0, v_0 = 0$
   for $j = 1, 2, 3, ..., m$
   $z = Av_j$
   $\alpha_j = v_j^T z$
   $z = z - \alpha_j v_j - \beta_{j-1} v_{j-1}$
   $\beta_j = ||z||_2$
   if $\beta_j = 0$, go to 4
   $v_{j+1} = z/\beta_j$
   end

4. Compute the eigenvalues of the symmetric tridiagonal matrix T_m consisting of α_j's and β_j's as shown in (1).

5. **If** convergence to all desired eigenvalues is obtained **then** quit **else** go to step 6.

6. Include the converged eigenvalue in the set of the fast Leja points by performing the Richardson iteration with the converged eigenvalues as shifts.

7. Update the interval $\mathbb{K}$ by including the part where the convergence is obtained. Compute new shifts in this updated interval and go to step 2.

We should know the input quantities before starting the algorithm. In section 5 we will discuss in detail how to estimate these quantities.

We saw earlier that Leja point generation is stable in a interval of length 4. Hence, we scale the spectrum of the given matrix to a interval of length 4. To achieve this we need the smallest and the largest eigenvalue of the matrix. We can estimate them by performing a few Lanczos iterations. It is also possible to scale the Leja points instead of the spectrum.

In step 2, we compute the fast Leja points in $\mathbb{K}$ which are used as shifts in Richardson iteration. This step provide us the desired starting vector for the Lanczos step.

If convergence to all desired eigenvalues is obtained then we stop, else we add the converged eigenvalues to the set of the fast Leja points. This means we perform Richardson iteration taking converged eigenvalues as shifts. Now, since the converged eigenvalue is a fast Leja point, we do not get convergence to it in the later runs.

For end eigenvalues it is easy to change the interval $\mathbb{K}$ after obtaining convergence to a desired eigenvalue. From our experiments we observed, extreme eigenvalues converge from one end of the wanted interval. Because of round-off error, orthogonalization of the starting vector against the converged Ritz vector is not enough to suppress the already converged eigenvalue in the later runs.

Now we move to the second algorithm which is a modified version of algorithm 1 to find the interior eigenvalues. Algorithm 2 differ from algorithm 1 after step 5.

Algorithm 2:

1. *Input:* End points of the interval $\mathbb{K}$, number of fast Leja points k, number of Lanczos steps m, normalized starting vector q_1.

2. *Richardson iteration*
   for $j = 1, 2, 3, ..., k$
   Compute ζ_j fast Leja point in $\mathbb{K}$.
   Set $q_{j+1} = (A - \zeta_j I)q_j$
   end

3. *Lanczos run*
   $v_1 = q_k/\|q_k\|_2$
   $\beta_0 = 0, v_0 = 0$
   for $j = 1, 2, 3, ..., m$
   $z = Av_j$
   $\alpha_j = v_j^T z$
   $z = z - \alpha_j v_j - \beta_{j-1} v_{j-1}$
   $\beta_j = ||z||_2$
   if $\beta_j = 0$, go to 4
   $v_{j+1} = z/\beta_j$
   end

4. Compute the eigenvalues of the symmetric tridiagonal matrix T_m consisting of α_j's and β_j's as shown in (1).

5. **If** convergence to all desired eigenvalues is obtained **then** quit **else** go to step 6.

6. Extend the sequence of the fast Leja points by computing more new shifts.

7. Orthogonalize the vector obtained from step 2 against the Ritz vectors corresponding to the converged eigenvalues.

8. Go to step 2 and start Richardson iteration with the above vector.

Unlike end eigenvalues, interior eigenvalues do not follow any obvious pattern while converging. Because of this, it is very difficult to keep track of all the intervals where convergence is obtained. Hence, change of interval is not suitable for such situations. To avoid the reappearance of the converged eigenvalues we opted for the orthogonalization of the starting vector against the converged Ritz vectors. As soon as convergence to one eigenvalue is obtained we orthogonalize the starting vector against the Ritz vector corresponding to this converged eigenvalue. Hence the step 7 is performed at every step only after achieving a convergence to an eigenvalue.

This algorithm does not perform as well as the first algorithm. This is to be expected since interior eigenvalues are much harder to compute using a standard Lanczos algorithm. Clustered eigenvalues are especially hard to find.

The Lanczos algorithm with a single starting vector cannot detect multiple eigenvalues. One way to overcome such a situation is to work with a block matrix which has at least as many columns as the required multiplicity of eigenvalues. This lead us to the final variant which works fine even when distinct but clustered interior eigenvalues are sought.

Algorithm 3:

1. *Input:* End points of the interval $\mathbb{K}$, number of fast Leja points k, number of Lanczos steps m, orthogonal random block Q_1.

2. *Richardson iteration*
   for $j = 1, 2, 3, ..., k$
   Compute ζ_j fast Leja point in $\mathbb{K}$.
   $Q_{j+1} = (A - \zeta_j I)Q_j$
   if $mod(j,\ korth) = 0$ ($korth$ is defined in Section 5.3)
   Orthogonalize Q_{j+1} internally
   end
   end

3. *Block Lanczos*
   $R_0 = Q_k$
   for $j = 1, 2, 3, ..., m$
   $R_{j-1} = V_j B_{j-1}$ (QR factorization of R_{j-1})
   $R_j = AV_j - V_{j-1}B_{j-1}^T$
   $M_j = V_j^T R_j$
   $R_j = R_j - V_j M_j$
   end

4. Compute eigenvalues of the block symmetric tridiagonal matrix T_m consisting of M_j's and B_j's as described in (2).

5. **If** convergence to all eigenvalues is achieved **then** quit **else** go to step 6.

6. Lock the converged eigenvectors by orthogonalizing them against the starting block.

7. Take the directions where convergence is not achieved and form a starting block for a new run of Richardson iteration.

8. Compute more shifts keeping old fast Leja points and go to step 2.

We know if the multiplicity of an eigenvalue does not exceed the block size then the block algorithm will find all copies at the same step. So, Golub and Van Loan [11, pg. 487] suggests that the block dimension should be at least as large as the largest multiplicity of any sought-after eigenvalues. Whereas, Cullum and Willoughby [6, pg. 210] and Bai *et al.* [3, pg. 54] suggest that the dimension of the block should be larger than the number of eigenvalues wanted. We start with a subspace whose dimension is the same as the desired number of the eigenvalues.

Unlike the single vector variant, we have to keep the orthogonality in all the blocks for the Lanczos step. So we may need to orthogonalize the basis while performing the Richardson iteration with fast Leja points as shifts. A condition which should be maintained to obtain a full rank basis is discussed in section 5.

When working with the block variant of Lanczos some eigenvalues converge faster than the others. As soon as we get desired accuracy in a wanted eigenvalue we lock the corresponding eigenvector, as discussed in Bai *et al.* [3, pg. 54]. In later steps, the size of the block is reduced by the number of converged eigenvalues. To avoid convergence to already converged eigenvalues we orthogonalize the new starting block against all the Ritz vectors corresponding to the converged eigenvalues.

5 Implementation Details

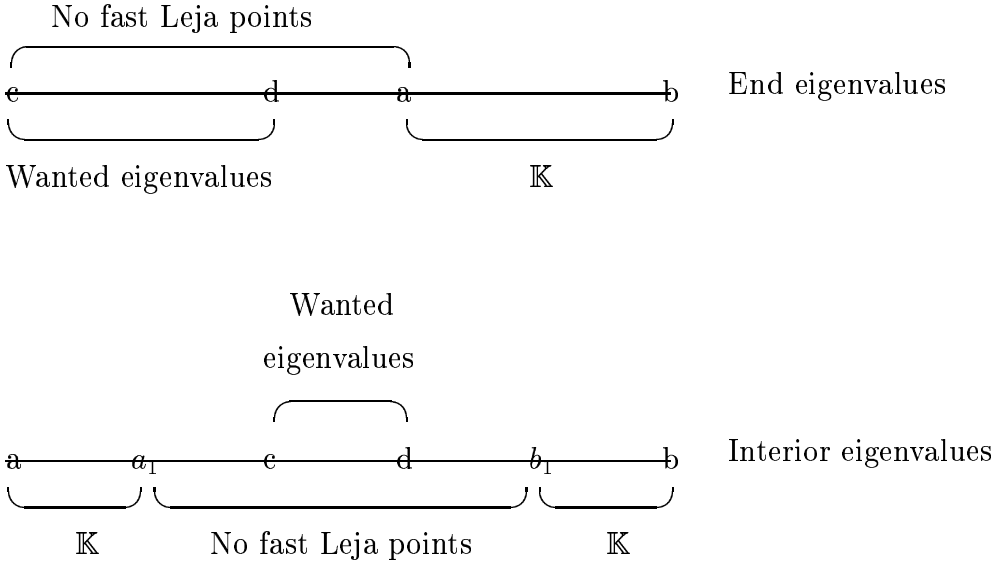
In this section we will discuss how to estimate the input quantities for the above three algorithms.

5.1 Number of fast Leja points

The most crucial input quantity is to approximate the number of the fast Leja points required to suppress the unwanted directions in the starting vector.

There are two ways to combine Leja points generation and the Richardson iteration. The first is, compute a long sequence of the fast Leja points and then use them as shifts in Richardson iteration. Hopefully in the end it will provide a good starting vector for the Lanczos step. The second approach is, performing the computation of fast Leja points and the Richardson iteration simultaneously, and stop when the undesired directions in a vector become small enough as compare to others. Among these two, the latter approach seem more obvious. We have only tested the first approach since it is easy to decide before hand approximately how many fast Leja points will be needed.

Our estimation is based on the Chebyshev polynomial in the interval $\mathbb{K}$. We saw in figure 3 that the zeros of Chebyshev polynomial and the fast Leja points are distributed almost identically in a same interval. The steep rise of the Chebyshev polynomial outside the interval $[-1, 1]$ is a boon. We have two situations to deal with according to the wanted interval. Pictorially they can be represented as



In the first case, we compute the Chebyshev polynomial over the interval $[a, b]$ and in the second case we work with the product of two Chebyshev polynomial in the intervals $[a, a_1]$ and $[b_1, b]$. The Chebyshev polynomial

over the interval $[a, b]$ is defined for λ outside the interval as

$$T_k(\lambda; [a, b]) \equiv T_k\left(1 + \frac{2(\lambda - b)}{b - a}\right) = \cosh\left(k\cosh^{-1}\left(1 + \frac{2(\lambda - b)}{b - a}\right)\right)$$

where k is the number of fast Leja points. When the desired interval lies inside the spectrum of the matrix, we work with $T_k(\lambda; [a, a_1]) \times T_k(\lambda; [b_1, b])$ and monitor the rise of the polynomial in the interval $[a_1, b_1]$.

Suppose the number of fast Leja points $k = k_1 + k_2$ where k_1 points are in $[a, a_1]$ and k_2 are in $[b_1, b]$. Let $y = 2(c - a_1)/(a_1 - a)$. Then

$$T_{k_1}(1 + y) = \frac{e^{-k_1 z} + e^{k_1 z}}{2}, \quad \text{where} \quad z = \cosh^{-1}(1 + y).$$

For small y we may approximate

$$\cosh^{-1}(1 + y) \approx log(1 + y + \sqrt{y(y + 2)})$$

For large k_1, $e^{-k_1 z} \to 0$. Hence,

$$T_{k_1}(1 + y) \approx \frac{e^{k_1 z}}{2} = \frac{\left(1 + y + \sqrt{y(y + 2)}\right)^{k_1}}{2}$$

Let tol be the required accuracy in the computed eigenvalues. Then

$$\begin{aligned} T_{k_1}(1 + y) &\geq 1/tol \\ \Rightarrow k_1 &\geq \frac{-\log(tol) + \log(2)}{\log(1 + y + \sqrt{y(y + 2)})} \end{aligned} \qquad (7)$$

Similarly, we approximate k_2 using $T_{k_2}(d; [b_1, b])$.

From (7), we observe that number of fast Leja points depends on the interval $[a_1, b_1]$ with respect to $[c, d]$. If c and d are very close to a_1 and b_1 respectively then we need more fast Leja points so that the Chebyshev polynomial become larger than the $1/tol$ in the interval $[c, d]$. In practice, we always work with the Leja polynomial. As soon as Leja polynomial at the point c becomes larger than $1/tol$ we need at least that many fast Leja points.

5.2 Interval $\mathbb{K}$ and number of Lanczos steps

As seen above, the interval $\mathbb{K}$ plays a very important role in approximating the number of fast Leja points. Too large or too small $\mathbb{K}$ can lead to extra computational effort. The length of $\mathbb{K}$ strongly depends on the gap between the wanted eigenvalues. Moreover, the length of the interval $\mathbb{K}$ and the spread of the interested eigenvalues play a very important role in deciding the number of Lanczos steps.

If the chosen $\mathbb{K}$ is too small then the interval where no fast Leja points are computed has many unwanted eigenvalues along with the wanted eigenvalues. So it is very likely that we may get large components along the directions corresponding to unwanted eigenvalues. In this way, we not only waste our effort on an uninterested quantity but also increase the computational cost by increasing the number of Lanczos steps.

There is a trade-off between the interval $\mathbb{K}$ and the number of Lanczos steps. If $\mathbb{K}$ is too large then we need many fast Leja points and very few Lanczos steps. And if $\mathbb{K}$ is too small than Lanzcos steps start to dominate the computational cost.

Suppose all eigenvalues of the matrix A are evenly distributed and there are m eigenvalues in the wanted interval $[c, d]$. If

$$\tau = \frac{\text{length}([a_1, b_1])}{\text{length}([c, d])}$$

then in $[a_1, b_1]$ we expect to have at least $m\tau$ eigenvalues. Hence, when the size of the starting block is $n \times m$ we need at most τ Lanczos steps to get the convergence to all $m\tau$ eigenvalues. We get better accuracy for wanted eigenvalues because the starting block has largest components in the direction of the wanted eigenvectors. This means, we need at least one step of Lanczos but never more than the ratio τ.

Hence, we decide the interval $\mathbb{K}$ so that the total computational cost which comprises of the Richardson iteration and the Lanczos iteration, is minimized. In our next section, we show an example that the interval $\mathbb{K}$ cannot be very small. There is some minimum number of fast Leja points which are required even if the interval where no fast Leja points are computed

is very large. So when we decrease the length of $\mathbb{K}$ we increase the number of the Lanczos steps. So we observe that after some time the total requirement of the fast Leja points in each interval remains the same but number of Lanczos steps increases.

5.3 Orthogonality

When we apply the Richardson iteration

$$Q_{k+1} = (A - \zeta_k I)Q_k, \quad k = 1, 2, ...$$

the basis Q_{k+1} should be linearly independent. Due to round-off, as k grows, the columns of Q_{k+1} tend to become parallel. To get a convergence to all the distinct eigenvalues we need to keep the vectors linearly independent. We do this by orthogonalizing the basis Q_k. To minimize the orthogonalization cost, we try to do it as seldom as possible.

There are many ways to measure the loss of orthogonality among the vectors obtained from the Richardson iteration. The linear independence of the vectors is best measured by the smallest singular value of Q_k. If the smallest singular value is much smaller than 1 then the vectors are becoming linearly dependent. In that case we do one step of orthogonalization. The singular values of Q_k are best measure but not cost effective. We are required to compute the singular values of a large non-square matrix at every Richardson step or at least after every few steps.

Another approach to measure the linear independence among the vectors of Q_k is QR with column pivoting. Let $\tilde{r}$ is a ratio between the maximum and minimum absolute value of the diagonal of R and g is a growth in $\tilde{r}$ after some steps of Richardson iterations. If $g \geq 1/tol$ then one step of orthogonalization is performed. We start with $g = 1$ and perform one step of QR-factorization of Q_k after some steps of Richardson iterations. We do not have a good guess when to orthogonalize for the first time. Later we orthogonalize only when the need arise. The benefit with this approach is that it is time saving. Also, provides a complete remedy in the sense that it checks and provide the orthogonal vectors simultaneously.

In the numerical experiment we compared this two approach for checking the orthogonality. We observed that after the first step of orthog-

onalization, the predicted growth from QR gives a very good indication for the loss of orthogonality and behaves similarly to the result obtained from the singular values of Q_k.

We can also measure the loss of orthogonality among the vectors by looking at the Leja polynomials. We took the idea from Rutishauser[17], where approximate eigenvalues of a tridiagonal matrix are used to check the orthogonality among the Lanczos vectors. This is done by looking at the maximum absolute value of the Leja polynomial over the interesting part of the spectrum and the maximum of the absolute value of the Leja polynomial at the left and right endpoint of the interesting interval $[c, d]$. We have observed if

$$\frac{\max(p(c), p(d))}{\max_{\zeta \in \mathbb{K}}(p(\zeta))} < 1/tol$$

then we do not need to orthogonalize Q_k.

The distances between c and a_1 and d and b_1 plays a very important role. If there are many eigenvalues in the interval $[a_1, c)$ and $(d, b_1]$ then these eigenvalues will influence the frequency of the orthogonality steps required. The problem in this approach root to the fact that we do not know how many eigenvalues are there in $[a_1, c)$ and what is the smallest eigenvalue in this interval.

From our experiments, shown in Section 6, we observed that if $[a_1, b_1]$ is much larger than $[c, d]$ then convergence to all the wanted eigenvalues is obtained long before the basis loses linear independence. Hence, we need to orthogonalize the starting block very seldom. We gain nothing by performing orthogonalization when the growth $g \geq 1/tol$.

5.4 Stopping criteria

The algorithm is stopped when user defined accuracy, *tol*, is obtained for all the desired eigenvalues. We compute the residual of the Ritz pair

$$r_i = Ax_i - x_i\theta_i = AV_j s_i - V_j s_i \theta_i = (AV_j - V_j T_j)s_i = v_{j+1}\beta_j e_j^T s_i.$$

If

$$||r_i||_2 = |\beta_j s_{ji}| \leq tol$$

then Ritz value θ_i is a good approximation of the eigenvalue λ_i of the matrix A.

In the case of the block variant,

$$||r_i||_2 = ||B_j(E_j^* s_i)||_2 \leq tol$$

where $E_j^* = (0, ..., 0, I_p)$ and $n \times p$ is the size of the starting block.

6 Numerical experiments

In this section we present the result of numerical experiments which illustrate the behaviour of the single vector and the block variant algorithms. We have used three sets of test matrices; namely, a diagonal matrix, the Anderson matrix and a Hamiltonian matrix.

The choice of diagonal matrices is based on the spread of eigenvalues. We consider two different diagonal matrices. If $A = diag(a_1, a_2, ..., a_n)$ then in first example $a_i = i$ where $i = 1, 2, ..., n$ and in the second case $a_i = i^2/n, i = 1, 2, ..., n$.

The second example comes from the Anderson model of localization and we call it the Anderson matrix [9]. These matrices come from quantum physics and the model is used for the investigation of electronic properties of disordered systems. The model for the matrix is taken from Elsner *et al.* [9] and is defined as follows:
The off-diagonal elements of a matrix obtained from Anderson model are equal to the off-diagonal elements of the 7-points central difference approximation to the three-dimensional Poisson equation. The matrices differ from each other in the diagonal entries, which are suitably chosen random numbers. In all our experiments diagonal entries are uniformly distributed random numbers in the interval [-1, 1]. Elsner *et al.*[9] made extensive experiments and found that the Cullum and Willoughby method [6] is best for computing eigenvalues for these matrices. The matrix is obtained from a $10 \times 10 \times 10$ grid and is of size $n = 1000$. Figure 5 shows a part of the spectrum.

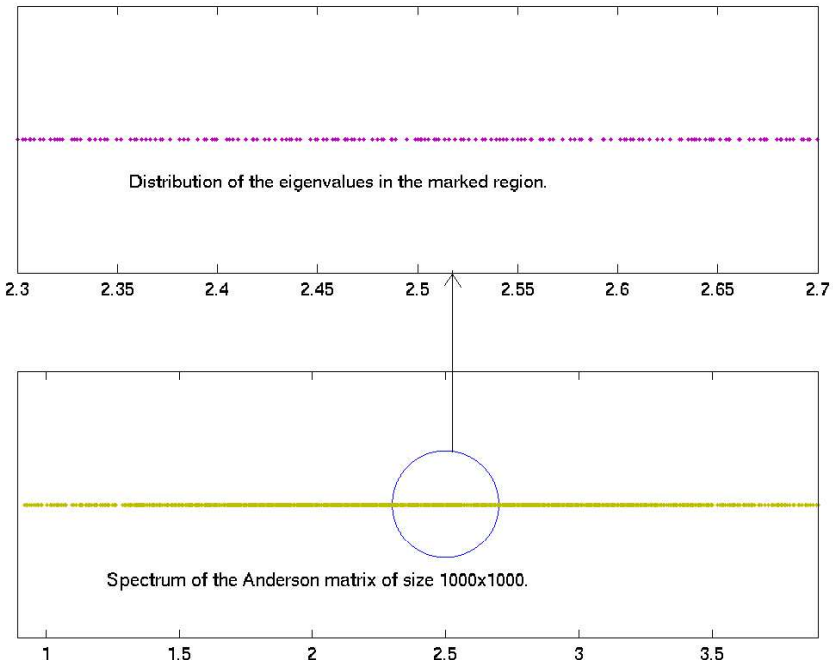


Figure 5: Spectrum of the 1000×1000 Anderson matrix.

The third set of test matrices are Hamiltonian matrices which arise in physical chemistry while studying the molecular excited state. A very simple model for these matrices is given by Wyatt in [19, 18]. These matrices consists of n_b blocks each of size n_s. Elements are defined as follows:
diagonals entries

$$(H_{i,i})_{j,j} = (i-1)\Delta + (j-1)\delta, \quad \delta << \Delta,$$

off-diagonal entries of diagonal block

$$(H_{i,i'})_{j,j} = C \exp(-|j-j'|),$$

entries of off-diagonal block

$$(H_{i,i'})_{j,j'} = [C/(n_{od}|i-i'|+1)] \exp(-|j-j'|).$$

where $i = 1, 2, ..., n_b$, $j = 1, 2, ..., n_s$, $i \neq i'$ and $j \neq j'$. $(H_{i,i'})_{j,j'}$ means jj'th entry of ii'th block. In all our runs $\Delta = 0.1, \delta = 10^{-4}, n_{od} = 5, C = 0.04$,

$n_b = 10$ and $n_s = 100$. Part of the spectrum of this matrix is plotted in figure 6.

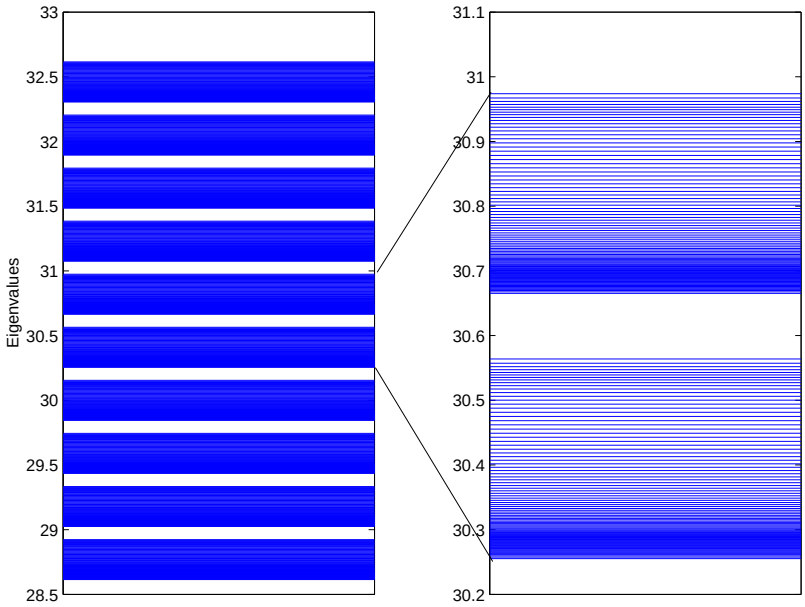


Figure 6: Eigenvalues for the model Hamiltonian matrix.

We will compute a few of the smallest and a few interior eigenvalues of all the above examples. The performance of algorithm 1 i.e the single vector variant for computing the end eigenvalues will be compared with the block algorithm described in algorithm 3, Cullum and Willoughby's no orthogonalization Lanczos and selective orthogonalization Lanczos. The block variant is used to find interior eigenvalues and is compared only with the Cullum and Willoughby's no orthogonalization Lanczos algorithm. In block variant algorithm the size of the starting block is same as the number of the wanted eigenvalues in all the examples.

For these examples, when employing either variant of the algorithm, we will discuss how to estimate the end points of $\mathbb{K}$, the number of fast Leja points and the number of Lanczos steps. For the block variant, we will even see when the need of a orthogonalization of the basis block in the Richardson iteration arises.

In figure 7, we show wanted eigenvalues and the corresponding gap. For the single vector variant, we are always considering the 10 smallest eigenvalues. Choice of interior eigenvalues is based on the distance in the eigenvalues.

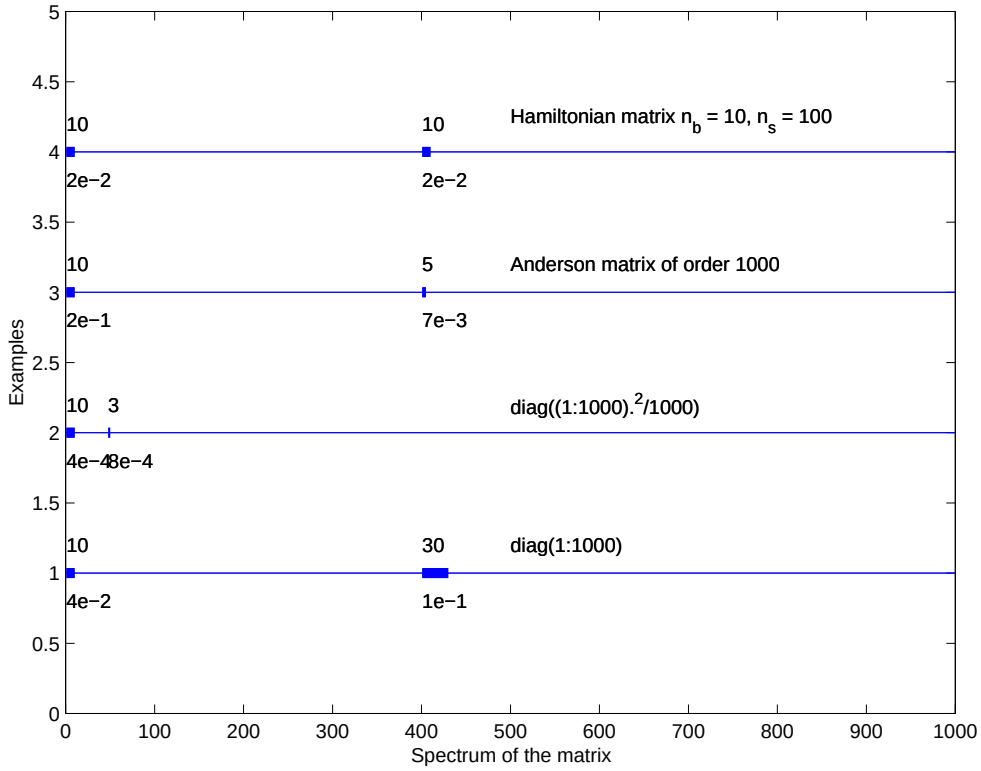


Figure 7: Desired eigenvalues in various test examples. Quantities above and below the line are number of wanted eigenvalues and the gap between them respectively.

The first step is to decide the required number of fast Leja points which will be needed to suppress all the directions in the interval $\mathbb{K}$. We will approximate the number of fast Leja points by plotting the Leja polynomial. We plot the Leja polynomial of different degrees and when the condition discussed in Section 5 is satisfied we take that many fast Leja points as shifts in the Richardson iterations. In other words, as soon as we reach the estimated degree, we take that many fast Leja points while computing the eigenvalues. In figure 8 we show the Leja polynomial for the end eigenvalues for all the examples which are amplified in the wanted spectrum.

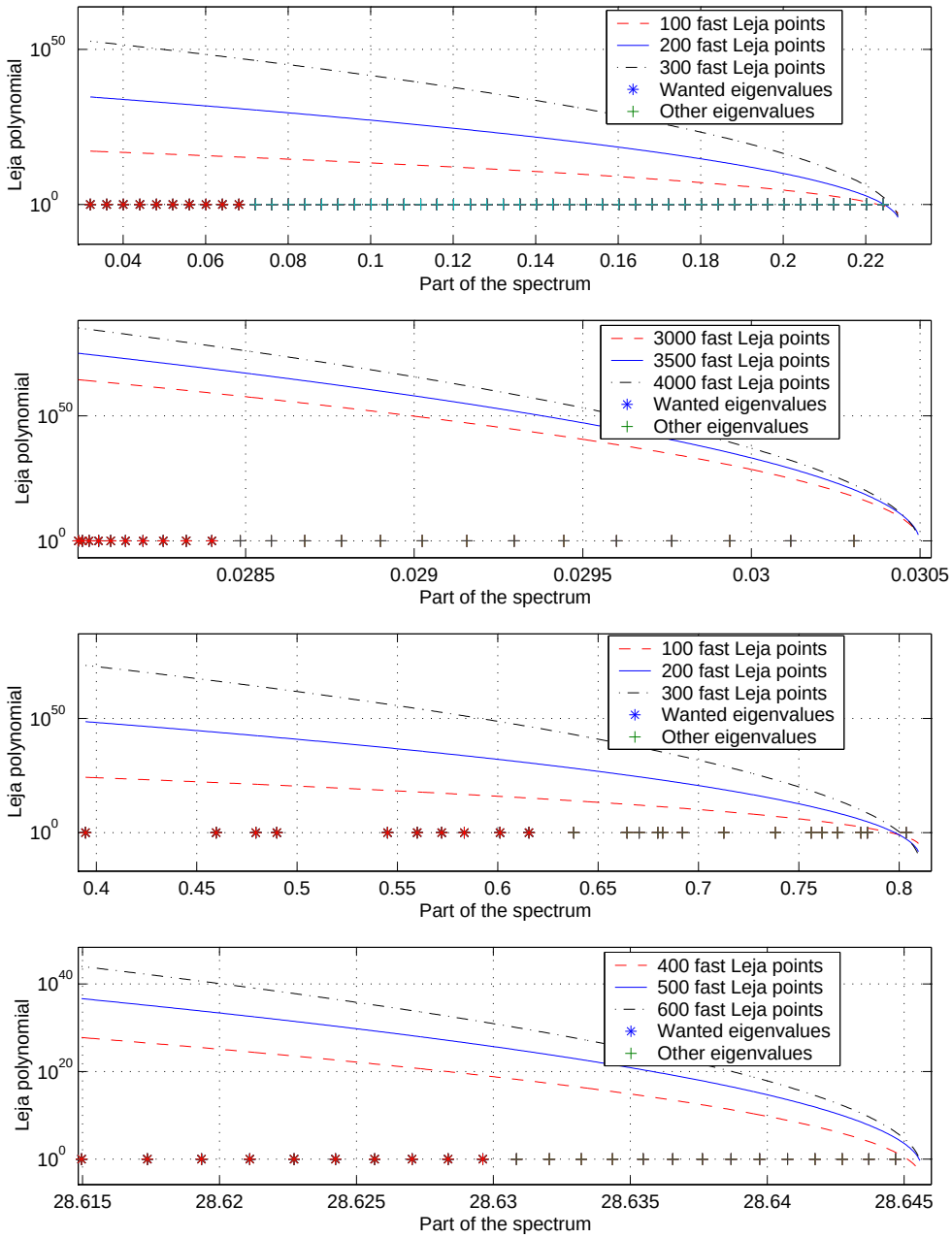


Figure 8: Leja polynomial in the interval where no fast Leja points are computed for the end eigenvalues. Examples from top to bottom: diag(1:1000), diag$(1:1000).^2/1000$, Anderson matrix, Hamiltonian matrix.

Similarly, in figure 9 we plot the Leja polynomial for the interior eigenvalues in the desired spectrum.

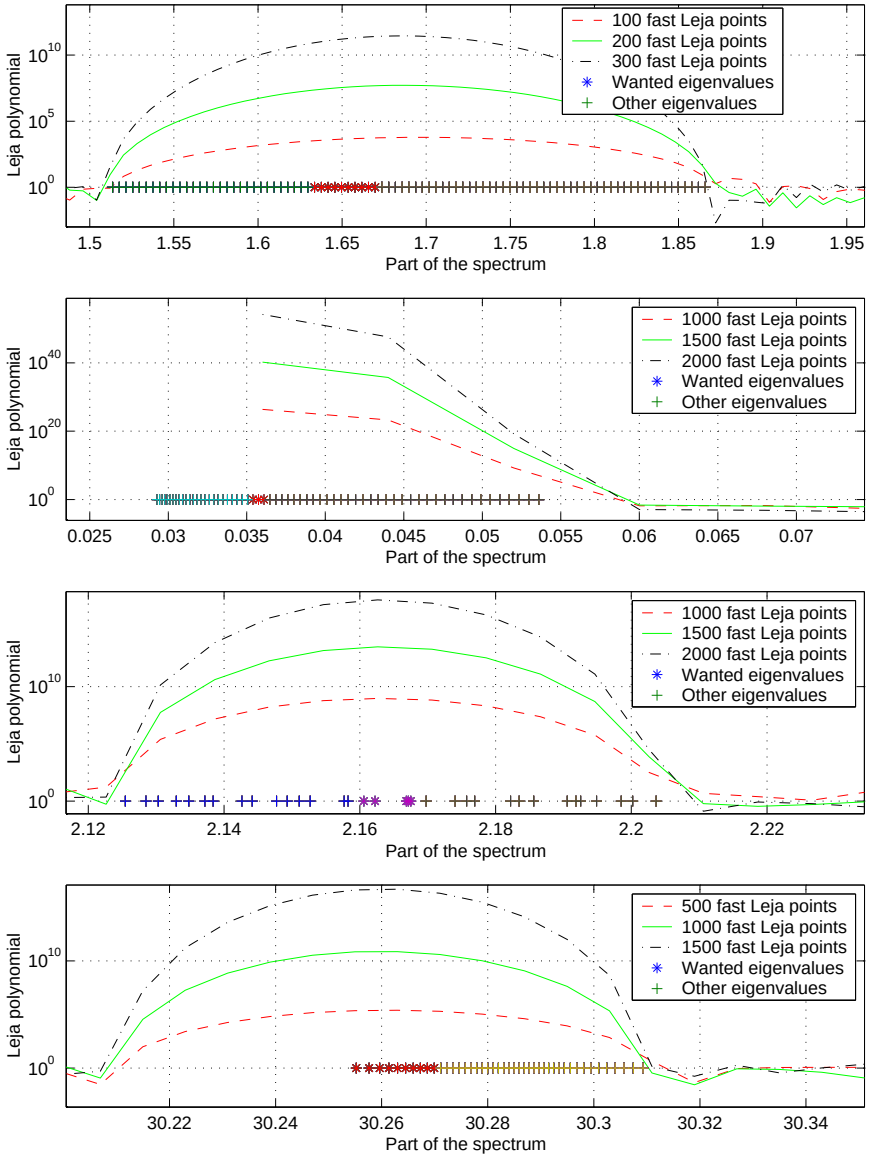


Figure 9: Leja polynomial in the interval where no fast Leja points are computed for the interior eigenvalues. Examples from top to bottom: diag(1:1000), diag$(1:1000).^2/1000$, Anderson matrix, Hamiltonian matrix.

In all the examples the interval where the desired eigenvalues lie is very small as compare to the full spectrum. In table 1 we give the description of the interval where no fast Leja points will be computed in the case of the end and the interior eigenvalues.

TABLE 1 Length of different intervals

Ex.	*Spectrum*	*Length of* $[a_1, b_1]$	
	$[a, b]$	*End*	*Interior*
1	[0.0320, 4.0320]	[0.0320, 0.2282]	[1.5095, 1.8699]
2	[0.0280, 4.0280]	[0.0280, 0.0305]	[0.0293, 0.0536]
3	[0.3946, 4.3946]	[0.3946, 0.8104]	[2.1236, 2.2045]
4	[28.615, 32.615]	[28.615, 28.646]	[30.21, 30.31]

From this table we observe that there are many eigenvalues in the interval $\mathbb{K}$. We get an estimate on the required number of fast Leja points from figures 8 and 9 where wanted interval $[c, d]$ is also shown. In case of the interior eigenvalues $[c, d]$ is in the middle of the interval $[a_1, b_1]$. These fast Leja points in $\mathbb{K}$ helps in amplifying the directions corresponding to the wanted eigenvalues by suppressing the unwanted directions corresponding to the eigenvalues in $\mathbb{K}$. The interval $[a_1, b_1]$ was decided so that the total number of fast Leja points and the number of Lanczos steps can be optimized. A smaller interval required more fast Leja points and larger was demanding more Lanczos steps. In figures 10a and 10b, we show the ratio between the interval where eigenvalues are desired and the interval where no fast Leja points will be computed in case of the end and the interior eigenvalues.

From this figure we also get an estimate for the required number of Lanczos steps. If the wanted eigenvalues are clustered together then the interval is large as compare to when they are nicely separated.

From the figures 8 and 9 we get the approximation on the required number of the fast Leja points. We compile the estimate number of fast Leja points in table 2 for the end and the interior eigenvalues. This will help us to compare between the estimate and the actual number of fast Leja points needed to get the convergence to all the eigenvalues. The entire algorithm is dominated by the matrix-vector multiplication. So it is very important that we do not use too many fast Leja points. Because this will not only increase the computational cost but also the computational time.

Figure 10a : End eigenvalues

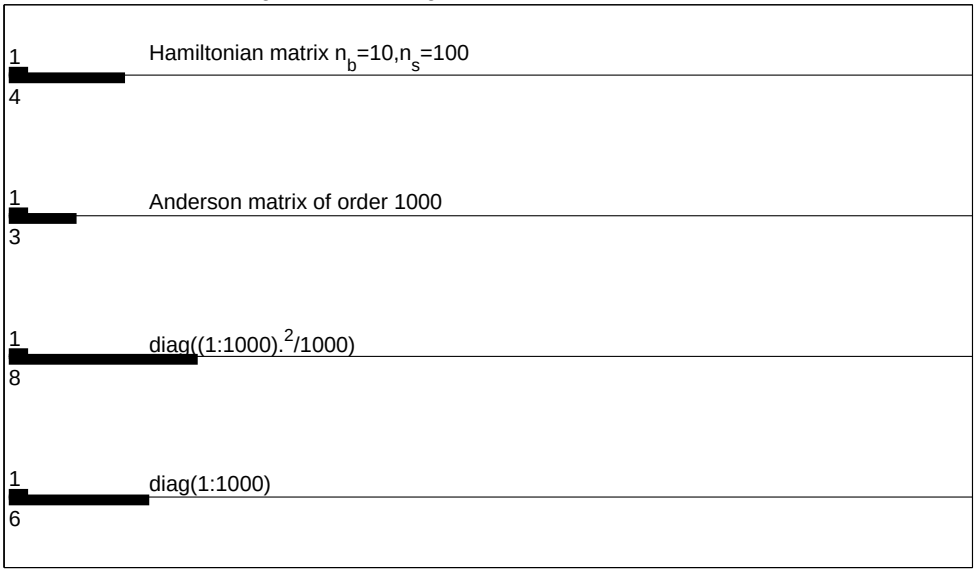


Figure 10b : Interior eigenvalues

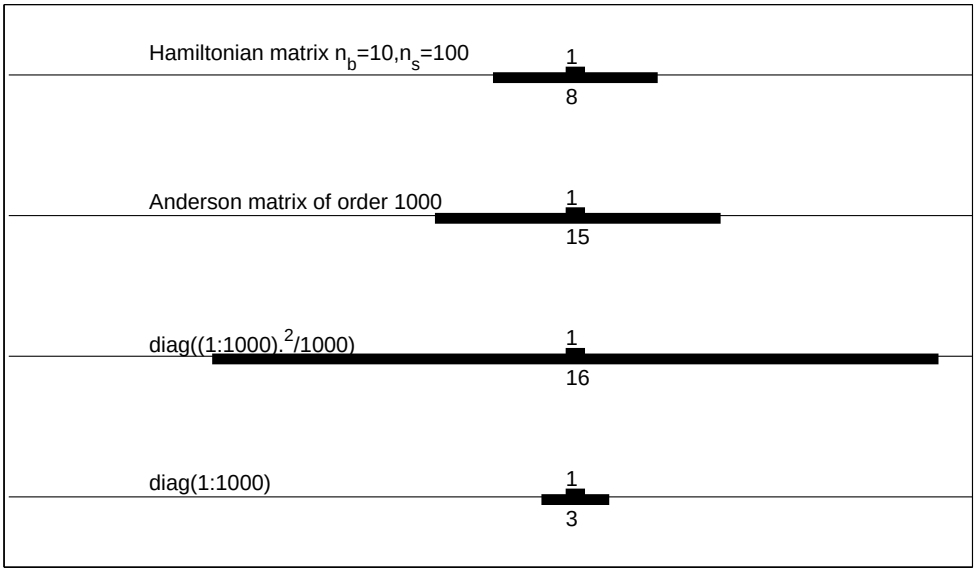


Figure 10: Estimation of the length of the interval where no fast Leja points will be computed.

TABLE 2 Estimation of fast Leja points

Ex.	*End eigenvalues*		*Interior Eigenvalues*
	Single	*Block*	
1	< 300	< 150	< 300
2	< 3500	< 1500	< 1500
3	< 150	< 100	< 1500
4	< 500	< 250	< 1500

Now we have all the input quantities for the algorithms 1 and 3. With this we are ready to implement the single and the block variant of the algorithm. In the block variant case the size of the starting block is $n \times m$ where n is the size of the matrix and m is the required number of wanted eigenvalues.

TABLE 3(a) Computation of end eigenvalues

Ex.	*Interval* $\mathbb{K}$	*Ratio* τ	*# of fast Leja points k*	*# of the Lanczos steps m*
1	$[\lambda_{50}, \lambda_n]$	5.4	200	20
2	$[\lambda_{25}, \lambda_n]$	6	2900	30
3	$[\lambda_{25}, \lambda_n]$	3	150	20
4	$[\lambda_{25}, \lambda_n]$	3	400	30

TABLE 3(b) Computation of interior eigenvalues

Ex.	*Interval* $\mathbb{K}$	*Ratio* τ	*# of fast Leja points k*	*# of the Lanczos steps m*
1	$[\lambda_1, \lambda_{370}] \cup [\lambda_{460}, \lambda_n]$	3	250	3
2	$[\lambda_1, \lambda_{18}] \cup [\lambda_{80}, \lambda_n]$	16	1500	11
3	$[\lambda_1, \lambda_{385}] \cup [\lambda_{420}, \lambda_n]$	15	1700	9
4	$[\lambda_1, 30.21] \cup [30.31, \lambda_n]$	7	1000	7

In the above two tables, table 3(a) and 3(b), we have compiled the results for the end and the interior eigenvalues respectively. These tables contains the optimum interval $\mathbb{K}$ and the ratio of the interval $[a_1, b_1]$ and $[c, d]$ for the respective examples. These tables also show the maximum number of the fast Leja points and the Lanczos steps needed to get the convergence

to all the wanted eigenvalues. From above tables we observe that results discussed in Section 5 give good estimate to the input quantities required for the algorithms discussed in Section 4.

Now we will compare the single vector variant and the block variant with the Cullum and Willoughby no reorthogonalization (C&W) and selective reorthogonalization (SO). In table 4(a) we compare methods for computing the smallest 10 eigenvalues. The entry in each block corresponds to the number of the fast Leja points, size of the tridiagonal matrix and the number of orthogonalization steps. Table 4(b) compare the result for interior eigenvalues.

TABLE 4(a) End eigenvalues- Comparison with other methods

Ex.	*Single vector*	*Block variant*	*C&W technique*	*Selective ortho.*
1	400+40+0	100+20+0	0+300+0	0+300+52
2	2900+30+0	300+10+0	0+1500+0	0+1000+1000
3	450+60+0	50+10+0	0+400+0	0+400+135
4	2800+210+0	300+100+0	0+1000+0	0+1000+ 996

TABLE 4(b) Interior eigenvalues- Comparison with other methods

Ex.	*Block variant*	*C&W technique*
1	250+90+0	0+1500+0
2	1500+33+0	0+3000+0
3	1700+45+0	0+6000+0
4	1000+70+0	0+10000+0

From table 4(a) we observe that the need of reorthogonalization of the Lanczos basis make selective orthogonalization algorithm expensive. Whereas, we never did any orthogonalization step in Cullum and Willoughby technique and block variant. As compare to Cullum and Willoghby technique, accuracy obtained in the computed eigenvalues was much better when approximated by the selective orthogonalization algorithm. In tables 5(a) and 5(b) we count explicitly the matrix-vector multiplication(MV) for each algorithm for the end and the interior eigenvalues.

TABLE 5(a) End eigenvalues- Comparison of matrix-vector multiplication

Ex.	*Single vector*	*Block variant*	*C&W*	*Sel. orth.*
1	440	1020	300	404
2	3100	3100	600	3000
3	510	510	400	670
4	3010	3100	1000	2992

TABLE 5(b) Interior eigenvalues-
Comparison of matrix-vector multiplication

Ex.	*Block variant*	*C&W*
1	2590	1500
2	15033	3000
3	17045	6000
4	10070	10000

Unlike Cullum and Willoughby method, we do not need to store any vector or matrix. Apart from storage the whole algorithm is dominated by Richardson iteration which consists of simple matrix-vector computation. Hence, the over all structure of the algorithm is not complicated.

We wrote all the algorithms i.e. 1, 2, 3 corresponding to the single and the block variant in MATLAB. The Cullum and Willoughby no reorthogonalization algorithm and the selective orthogonalization algorithm are written in Fortran 90 and MATLAB so as to handle large tridiagonal matrix and reorthogonalization efficiently. Because of this we didn't compare the total execution time taken by each algorithm.

In all the above examples we never performed a single step of orthogonalization in the Richardson iteration while implementing the block variant algorithm. The reason being the interval $[a_1, b_1]$ chosen is larger than the interval $[c, d]$. If $[a_1, b_1]$ and $[c, d]$ are same or almost same then we may require to perform few steps of orthogonalization to keep the basis linear independent to the working precision.

Let us consider a 1000×1000 diagonal matrix A whose diagonal elements are evenly distributed in the interval $[-2, 2]$. We take the interval $\mathbb{K}$ to be $[-2, -1] \cup [1, 2]$. We will consider two different intervals $[c, d]$ and

will compute all the eigenvalues within this interval. In the first case we will find all the 250 eigenvalues in the interval $[-0.5, 0.5]$ and in the second case interval under consideration will be $[-0.9, 0.9]$ which has 450 eigenvalues.

We get convergence to all the wanted 250 eigenvalues in $[-0.5, 0.5]$ with 40 fast Leja points, as seen from the figure 11.

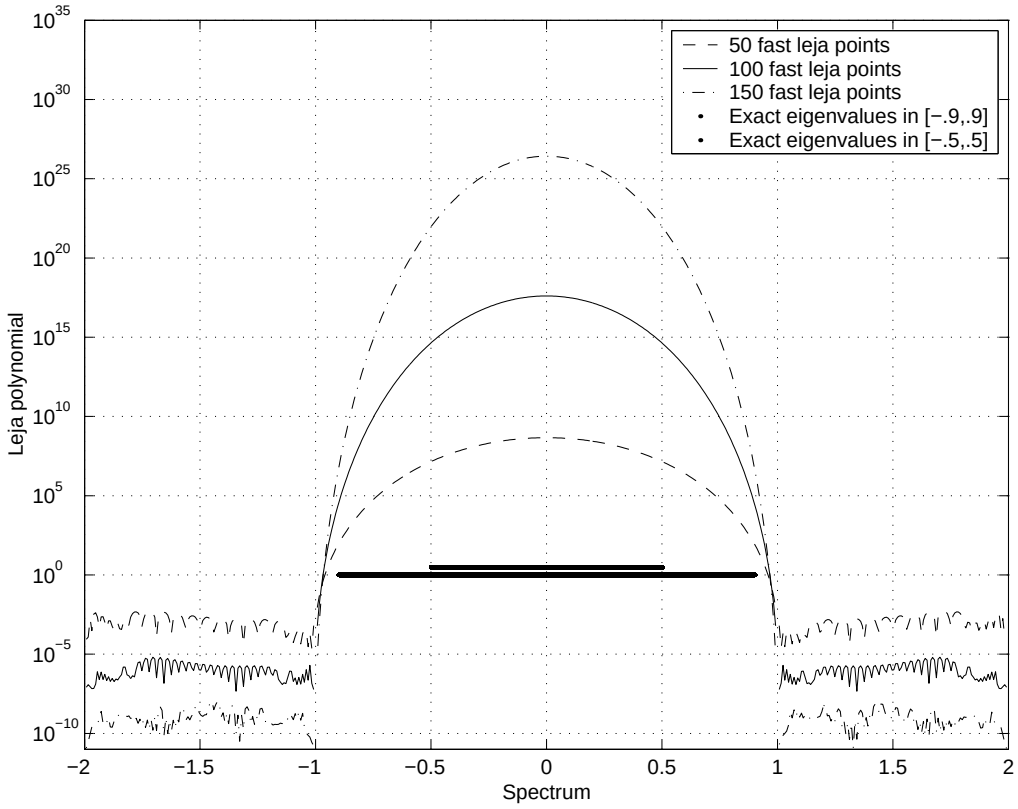


Figure 11: Leja polynomial of 1000x1000 diagonal matrix.

Since we forced one step of orthogonalization after 10 steps of Richardson iteration, we see in figure 12 that after that we do not require any more orthogonalization step. As said in Section 5.3, we start with growth $g = 1$ and orthogonalize after pre-assigned number of Richardson iterations. After the first QR-factorization we get the condition number of the basis by looking at $\tilde{r}$, defined in Section 5.3. From $\tilde{r}$ we estimate the condition number after one step of Richardson iteration. We keep a check on the growth of this condition number after every step of Richardson iteration and when it becomes larger

than $1/tol$ we orthogonalize the basis for the second time. We denote this growth by g which is updated at every step by multiplying with the condition number after one step of Richardson iteration.

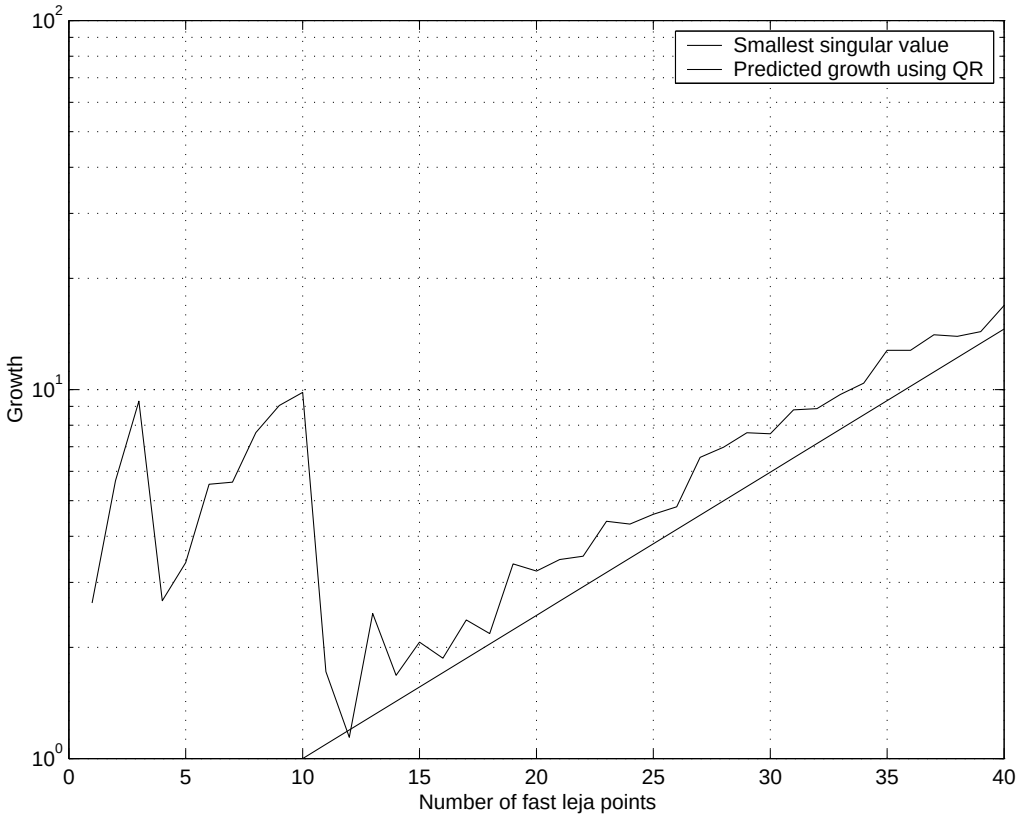


Figure 12: Checking loss of orthogonality when $[c, d] = [-0.5, 0.5]$. We force one step of orthogonalization after 10 steps.

But if we relax the first orthogonalization then we get convergence to all the eigenvalues without performing a single step of Richardson iteration. This is shown in following figure 13.

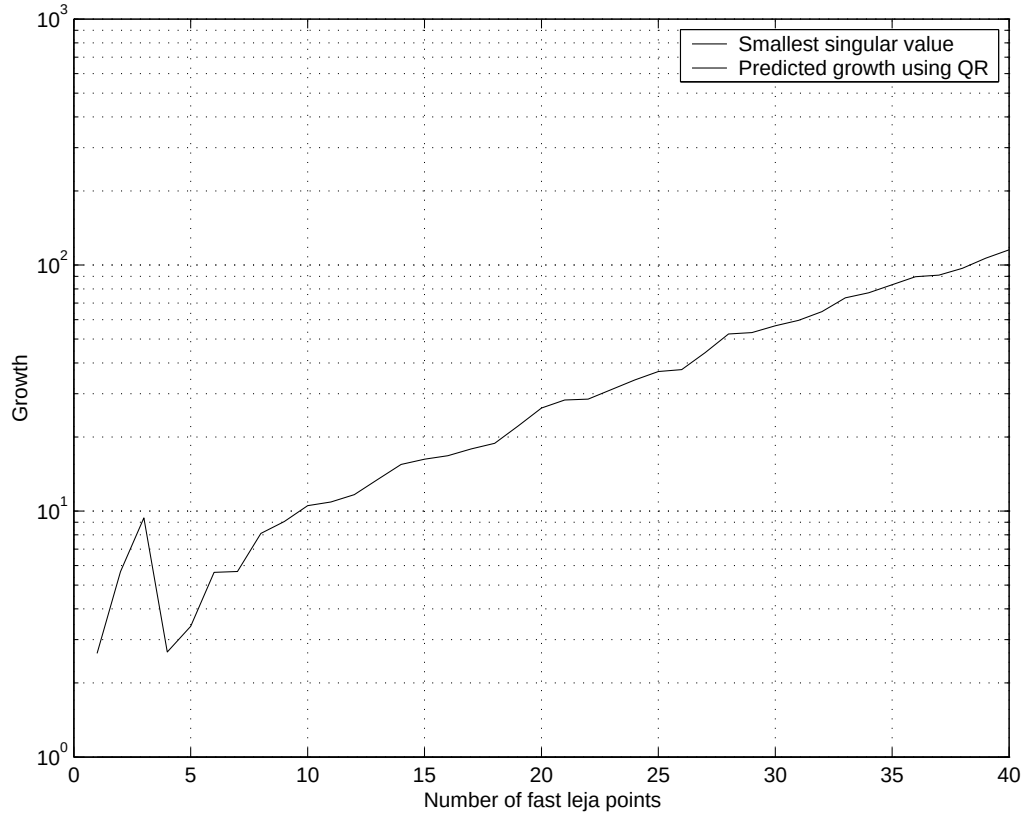


Figure 13: Checking loss of orthogonality when $[c, d] = [-0.5, 0.5]$. We do not force when to do the first orthogonalization.

Now lets consider the second interval $[-0.9, 0.9]$. From figure 11 we know approximately 100 fast Leja points are needed for the convergence. Figure 14 tells us that 2 steps of orthogonalization were performed.

The first one is after 10 steps and the other when the predicted growth become larger than the tolerance 10^8. If we orthogonalize when growth become larger than 10^4 then orthogonalization was performed 3 times to maintain the linear independence among the Richardson vectors. This is depicted in figure 15.

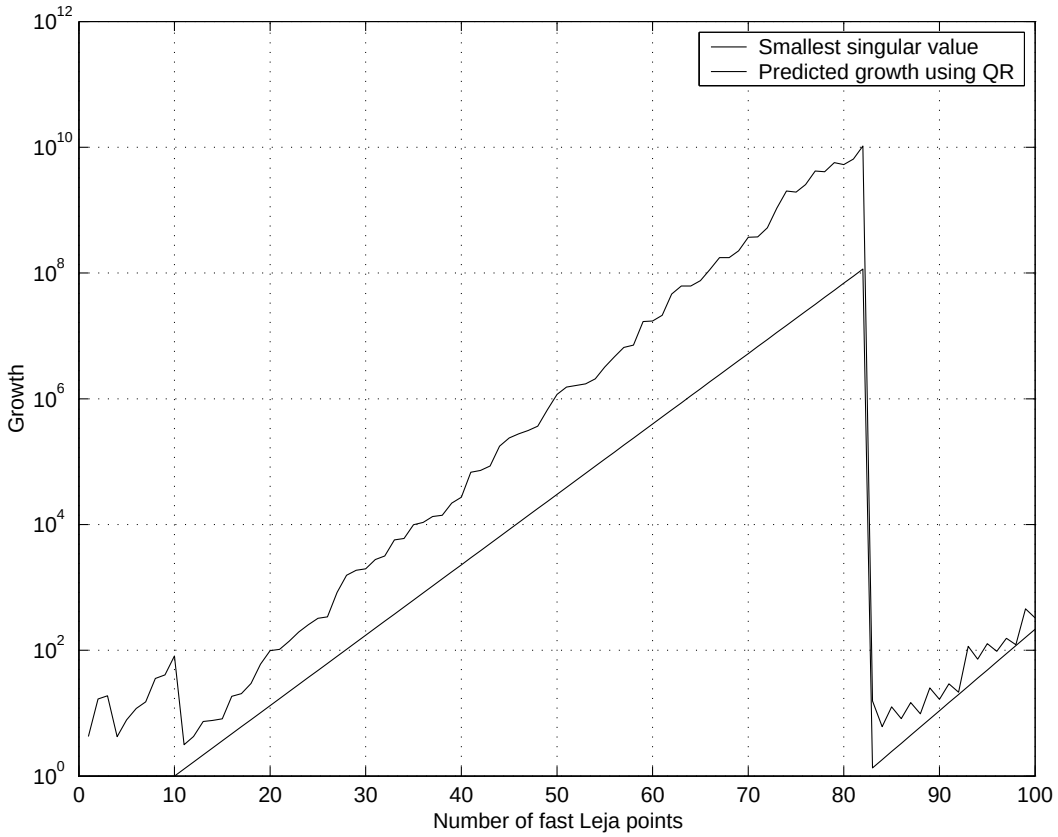


Figure 14: Checking loss of orthogonality when $[c, d] = [-0.9, 0.9]$. We orthogonalize when growth become larger than 10^8.

These figures hint at taking larger $[a_1, b_1]$ as compare to $[c, d]$. From these figures it is also noticeable that after the first step of orthogonalization, smallest singular value and the predicted growth obtained using the QR-factorization of the Richardson block behaves similarly. Hence, we can rely on this computed growth to monitor the loss of linearly independence. This will also avoid the computation of singular values at every step of Richardson iteration.

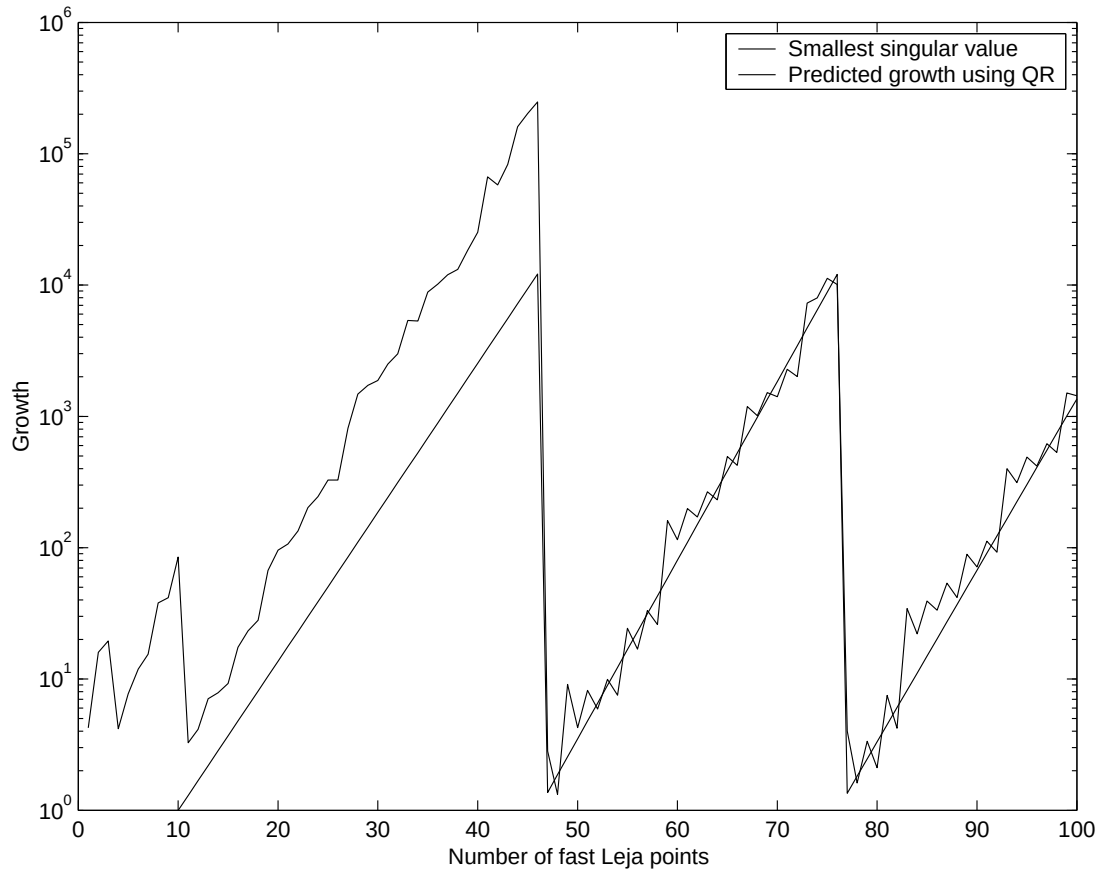
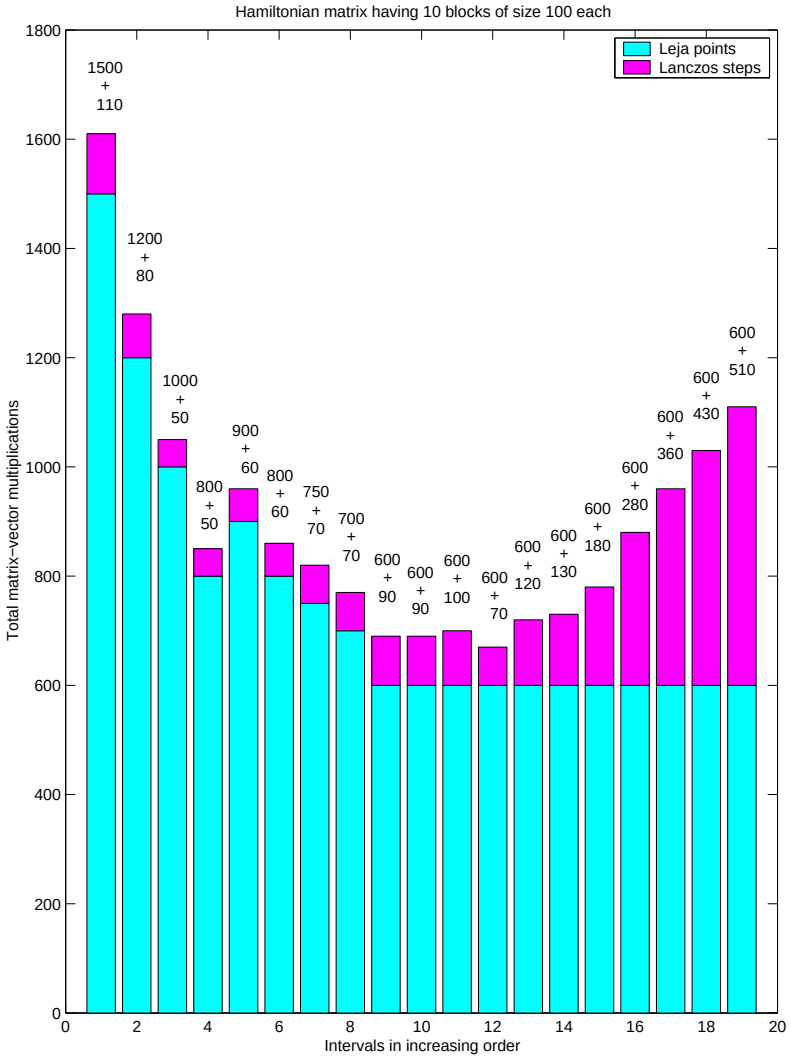


Figure 15: Checking loss of orthogonality when $[c, d] = [-0.9, 0.9]$. We orthogonalize when growth become larger than 10^4.

The other thing which is interesting to observe is the relation between the number of fast Leja points and the number of Lanczos steps, which we talked about in Section 5.2. Let $[c, d]$ be the wanted spectrum and $[a_1, b_1]$ be the interval where no fast Leja points will be computed. Then if length of $[a_1, b_1]$ is small then we need more fast Leja points and few Lanczos steps and if length of $[a_1, b_1]$ is large then we need less fast Leja points and more Lanczos steps. There is a trade off between fast Leja points and Lanczos iterations. In figure 16 we tried to show that if the length of $[a_1, b_1]$ is too big as compare to $[c, d]$ then it is Lanczos algorithm which starts to dominate the entire algorithm.



+

Figure 16: Comparison between the fast Leja points and the Lanczos iterations.

In the above figure, we are considering the 1000×1000 Hamiltonian matrix and we are interested in eigenvalues from 201 to 210. We start with small $[a_1, b_1]$ where Richardson iteration is more expensive because of many fast Leja points. But later when $[a_1, b_1]$ becomes considerably large than Lanczos

runs becomes dominating factor in the algorithm.

In the end, we tried our algorithm on some large sparse matrices again coming from previous 3 categories. Except for the diagonal case, we do not know the exact eigenvalues of the matrix. We estimate the smallest and the largest eigenvalue of the respective matrix by using the *eigs* command in MATLAB. We could have done this computation by doing few runs of Lanczos iteration. Once we know the end eigenvalues we scale the matrix so that all the eigenvalues lie in an interval of length 4. We decide the wanted interval $[c, d]$ and using the conditions from section 5 we estimate the interval $[a_1, b_1]$, number of fast Leja points, orthogonality condition and the number of Lanczos iterations. The algorithm is stopped when the condition discussed in section 5.5 is satisfied.

We restricted ourselves to finding the 10 interior eigenvalues using the block variant i.e. algorithm 3. The block size of the starting block is $n \times 10$ and accuracy desired in the wanted eigenvalues is less than or equal to 10^{-10}. We did not compare the results with Cullum and Willoughby no reorthogonalization because we are not having an efficient algorithm to find the eigenvalues of a large tridiagonal matrix. In the following table for each example we mention the size of the matrix, wanted interval $[c, d]$, interval $[a_1, b_1]$, number of fast Leja points and Lanczos iterations and the CPU time. The CPU time does not incur the initialization of the data and the computation of the fast Leja points. Hence, CPU time shows the time spend on the Richardson iteration and Lanczos iteration. We have run all the examples on a 330 MHz Sparc workstation and using the MATLAB 6.0 version.

TABLE 6 Results from large matrices

Ex.	*Size*	$[c, d]$	$[a_1, b_1]$	*nflp*	m	*CPU time*
1	10000	[2.0034,2.0070]	[1.93,2.08]	3000	27	92
1	20000	[2.0017,2.0035]	[1.94,2.06]	7000	30	501
1	40000	[2.0009,2.0018]	[1.991,2.01]	17000	15	2176
2	10000	[1.0032, 1.0068]	[0.965,1.042]	3000	25	113
2	20000	[1.0016,1.0034]	[0.981,1.023]	6000	34	570
2	40000	[1.0008,1.0017]	[0.985,1.016]	13000	25	2383
3	10648	[2.1608,2.1684]	[2.1356,2.1936]	17000	25	1346
3	21952	[2.1608,2.1684]	[2.1356,2.1936]	26000	25	2573
3	39304	[2.1608,2.1684]	[2.1356,2.1936]	47000	25	6358

In the above table example 1 means diag(1:n), example 2 corresponds to diag$(1 : n).^2/n$ and Anderson matrix is given by example 3. Also, number of fast Leja points is given by *nflp* and number of Lanczos steps by m. From this table it is evident that number of fast Leja points increase as the size of the matrix increase. Moreover, for the clustered eigenvalues work load is more. This fact is pronounced even in the number of the Lanczos steps. In all the example size of the block is $n \times 10$.

7 Conclusions

The bounds which we define in Section 5 to estimate the end points of the interval $\mathbb{K}$, number of fast Leja points k, orthogonality and the Lanczos steps m give reasonably good estimates. In some examples the estimate to the number of Lanczos steps is much larger than the actual number of steps required.

The Cullum and Willoughby method performs very well while computing the end eigenvalues. Reorthogonalization requires a substantial effort in selective orthogonalization technique. The work required by the block Lanczos is little more than the single vector variant.

While computing the interior eigenvalues we observe that when the spectrum of the matrix is dense, the Cullum and Willoughby method need

many Lanczos steps. This means we have to work with a very large tridiagonal matrix. Whereas with the block variant we need very short Lanczos runs. The required number of fast Leja points depend on the spread of the wanted eigenvalues. We observe that the total need of fast Leja points is large but the Richardson iteration consists of simple matrix vector multiplication.

There are few points which we would like to study further in more detail. In case of the small matrices we worked with the exact eigenvalues of the matrix and in case of the large matrices we estimated the end eigenvalues using the MATLAB build-in function *eigs*. We approximated the eigenvalues to the full accuracy i.e. when the residual is less than 10^{-10}. We would like to estimate the end eigenvalues by Lanczos algorithm and would like to investigate how accurately do we need to approximate the end eigenvalues. What will be the consequences if they are not accurate to the full machine accuracy say 10^{-10} or less? Since fast Leja points generation need end eigenvalues, this is an interesting question to probe further.

The next question is related to the Richardson iteration and the fast Leja points. We compute long sequence of the fast Leja points and then apply the Richardson iteration using them as shifts. If we do not get convergence then more fast Leja points are computed. As said in section 5.2, performing fast Leja points and Richardson iteration simultaneously seem more obvious. We would like to implement this approach and compare with the present version of the algorithm.

Among all the three algorithms we worked extensively with algorithms 1 and 3. We discarded the algorithm 2 on the argument that it is difficult to find interior eigenvalues using single vector Lanczos algorithm. We think it will be interesting to see how better algorithm 3 i.e. block variant performs as compare to the algorithm 2 which is a single vector Lanczos algorithm.

References

[1] J. Baglama, D. Calvetti, and L. Reichel. Iterative methods for the computation of a few eigenvalues of a large symmetric matrix. *BIT*, 36:400–421, 1996.

[2] J. Baglama, D. Calvetti, and L. Reichel. Fast Leja points. *ETNA*, 7:124–140, 1998.

[3] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. Van der Vorst. *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*. SIAM, Philadelphia, 2000.

[4] D. Calvetti and L. Reichel. An adaptive Richardson iteration method for indefinite linear systems. *Numerical Algorithms*, 12:125–149, 1996.

[5] D. Calvetti, L. Reichel, and D. C. Sorensen. An implicitly restarted Lanczos method for large symmetric eigenvalue problems. *ETNA*, 2:1–21, 1994.

[6] J. Cullum and R. A. Willoughby. *Lanczos Algorithms for Large Symmetric Eigenvalue Computations*, volume 1. Birkhauser, Basel, 1985.

[7] J. W. Demmel. *Applied Numerical Linear Algebra*. SIAM Publications, Philadelphia, PA, 1996.

[8] A. Edrei. Sur les déterminants récurrents et les singularités d'une fonction donnée par son developpement de Taylor. *Compositio Mathematica*, 7:20–88, 1939.

[9] U. Elsner, V. Mehrmann, F. Milde, R. A. Römer, and M. Schreiber. The Anderson model of localization: A challenge for modern eigenvalue methods. *SIAM J. Sci. Comput.*, 20:2089–2102, 1999.

[10] T. Ericsson and A. Ruhe. The spectral transformation Lanczos method for the numerical solution of large sparse generalized symmetric eigenvalue problems. *Math. Comp.*, 35:1251–1268, 1980.

[11] G. Golub and C. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, MD, 3rd edition, 1996.

[12] G. H. Golub and R. Underwood. The Block Lanczos method for computing eigenvalues. *Mathematical Software III*, pages 364–377, 1977.

[13] F. Leja. Sur certaines suites liées aux ensembles plans et leur application à la representation conforme. *Annales Polonici Mathematici*, 4:8–13, 1957.

[14] C. C. Paige. *The computation of eigenvalues and eigenvectors of very large sparse matrices.* PhD thesis, London University, 1971.

[15] B. Parlett. *The Symmetric Eigenvalue Problem.* Prentice Hall, Englewood Cliffs, NJ, 1980.

[16] L. Reichel. Newton interpolation at Leja points. *BIT*, 30:332–346, 1990.

[17] H. Rutishauser. Computational aspects of F. L. Bauer's simultaneous iteration method. *Numer. Math.*, 13:4–13, 1969.

[18] A. Vijay and R. E. Wyatt. Spectral filters in quantum mechanics: a mesurement theory perspective. *Physical Review E*, 62:4351–4364, 2000.

[19] R. E. Wyatt. Matrix spectroscopy: Computation of interior eigenstates of large matrices using layered iteration. *Physical Review E*, 51:3643–3658, 1995.

[20] D. M. Young. *Iterative Solution of Large Linear Systems.* Academic Press, Inc, New York, 1971.